University of Zurich[UZH]

Department of Informatics

Negation in Temporal-Probabilistic Databases

A dissertation submitted to the Faculty of Economics, Business Administration and Information Technology
of the University of Zurich

for the degree of
Doctor of Science (Ph.D.)

by

Aikaterini Papaioannou

from Athens, Greece

Accepted on the recommendation of

Prof. Dr. Michael H. Böhlen
Prof. Dr. Martin Theobald
Prof. Dr. Abraham Bernstein

2019

University of Zurich[UZH]

The Faculty of Economics, Business Administration and Information Technology of the University of Zurich herewith permits the publication of the aforementioned dissertation without expressing any opinion on the views contained therein.

Zurich, February 13th, 2019

Head of the Ph.D. committee for informatics: Prof. Dr. Thomas Fritz

Abstract

The need to manage large, temporal-probabilistic (TP) datasets appears in a wide range of applications such as data cleaning, scientific databases, RFID data. Computing algebra operations on interval timestamped relations with probabilities complies with the semantics of the temporal and the probabilistic dimension and lays in the manipulation of intervals and lineage expressions as well as in the computation of output probabilities. Despite the extensive research in temporal and probabilistic databases, the combination of intervals and probabilities has received little attention and the computation of TP operations with negation has not been covered. In this work we propose algorithms for the efficient computation of the output intervals and lineage expressions on these operations that can be integrated in the kernel of PostgreSQL and we also propose means to get useful insights on their results.

The solution we propose for TP operations with negation uses two novel mechanisms: the *lineage-aware temporal windows* and the *generalized lineage-aware temporal windows*, appropriate for the computation of TP set-operations, including TP set-difference, and for TP outer-joins and anti-join, respectively. The key feature of these windows is that they bind an output interval with the facts and the lineage-expressions of the input tuples that contribute to the output over the interval. This feature allows for alleviation of the expensive joins that are otherwise used to either perform the lineage-and-interval binding or to filter the output tuples.

Lineage-aware temporal windows comply with the requirement of set-operations to combine only tuples with the same non-temporal attributes. For their computation, we propose an algo-

rithm that exploits properties of the model and thus computes TP set-operations with linearithmic complexity, outperforming existing quadratic solutions. *Generalized lineage-aware temporal windows* are appropriate for TP outer-joins and TP anti-join where multiple tuples of each relation might be valid over an output interval and where input tuples with different non-temporal attributes might be combined to form an output tuple. On top of that, TP outer-joins combine the characteristics of TP operations with and without negation: at each time-point, two outcomes might arise based on the same valid tuples since they can be both $true$ and $false$. For the computation of TP outer-joins and TP anti-join, we group the *generalized lineage-aware temporal windows* of two TP relations into three disjoint sets and we propose algorithms that compute these sets incrementally, alleviating a lot of redundant computations and improving the runtime of TP joins with negation.

We implement our approach in the kernel on PostgreSQL and we introduce TemProRA, a system that is built on top of our PostgreSQL implementation. TemProRA illustrates the result of a TP operation and provides three visualization tools for the analysis and tuning of the k result tuples of a temporal-probabilistic query.

Zusammenfassung

In einer Vielzahl von Anwendungen wie Datenbereinigung, wissenschaftlicher Datenbanken und RFID Daten ist es notwendig, grosse,temporal-probabilistiche (TP) Datensätze zu verwalten. Die Berechnung von Algebraoperatoren auf Relationen mit Zeitintervallen und Wahrscheinlichkeiten genügt der Semantik der temporalen und der probabilistischen Dimension und basiert sowohl auf der Anpassung von Intervallen und Lineage-Ausdrücken als auch auf der Berechnung der Ausgabewahrscheinlichkeiten. Trotz der umfangreichen Forschung im Bereich temporaler und probabilistischer Datenbanken, hat die Kombination aus Intervallen und Wahrscheinlichkeiten wenig Aufmerksamkeit erhalten und die Berechnung von TP-Operationen mit Negation wurde nicht betrachtet. In dieser Arbeit schlagen wir Algorithmen für die effiziente Berechnung von Ausgabeintervallen und Lineage-Ausdrücken für diese Operatoren vor, die in PostgreSQL integriert werden können. Weiterhin zeigen wir Mittel auf, um relevante Einblicke in die Ergebnisse zu erhalten.

Unsere Lösung für TP-Operationen mit Negation verwendet zwei neue Mechanismen: die *lineage-aware temporal Windows* und die *verallgemeinerten lineage-aware temporal Windows*. Erstere eignen sich für die Berechnung von Mengenoperatoren wie TP-Mengendifferenz und letztere für die Berechnung von TP-Outer Joins und TP-Anti Joins. Das Hauptmerkmal dieser Windows ist, dass sie das Ausgabeintervall mit den Fakten und Lineage-Ausdrücken der Eingabetupel, die zur Ausgabe über dieses Intervall beitragen, verknüpfen. Dieses Merkmal ermöglicht

es, die teuren Joins zu vermeiden, die andernfalls verwendet werden müssten, um entweder die Verknüpfung von Lineages und Intervallen auszuführen oder um die Ausgabetupel zu filtern.

Lineage-aware temporal Windows erhalten die Eigenschaft von Mengenoperationen, nur Tupel mit den gleichen nicht-temporalen Attributen zu kombinieren. Für ihre Berechnung, schlagen wir einen Algorithmus vor, der die Eigenschaften des Modells ausnutzt und daher die TP-Mengenoperationen mit Komplexität $O(n \log n)$ berechnet, was eine Verbesserung gegenüber den existierenden Lösungen, die quadratische Komplexität haben, darstellt. *verallgemeinerte lineage-aware temporal windows* eignen sich für TP-Outer Joins und TP-Anti Joins, bei denen mehrere Tupel jeder Relation während eines Ausgabeintervalls gültig sein können und bei denen Tupel mit unterschiedlichen nicht-temporalen Attributen zu einem Ausgabetupel kombiniert werden können. Hinzu kommt, dass TP-Outer Joins die Eigenschaften von TP-Operationen mit und ohne Negation kombinieren: zu jedem Zeitpunkt können sich zwei Ergebnisse auf Grundlage derselben gültigen Tupel ergeben, da diese sowohl *wahr* als auch *falsch* sein können. Für die Berechnung von TP-Outer Joins und dem TP-Anti Join teilen wir die *verallgemeinerten lineage-aware temporal Windows* der zwei TP-Relationen in drei disjunkte Mengen auf and wir schlagen Algorithmen vor, die diese Mengen inkrementell berechnen und dadurch viele redundante Berechnungen vermeiden und die Laufzeit von TP-Joins mit Negation verbessern.

Wir implementieren unseren Ansatz in das Datenbanksystem PostgreSQL und führen TemProRA ein, ein System, das mit unserer PostgreSQL Implementierung interagiert. TemProRA veranschaulicht das Ergebnis einer TP-Operation und stellt drei Visualisierungswerkzeuge für die Analyse und Anpassung der Ergebnistupel einer temporal-probabilistischen Anfrage zur Verfügung.

To my partner, my parents, my brother
your love and support are the most precious
gifts I have ever been given

Acknowledgments

First and foremost, I would like to thank my advisor Prof. Dr. Michael Böhlen for giving me an opportunity to pursue a PhD in the Database Technology Group (DBTG). I am grateful for his guidance and insightful feedback. His consistency is a virtue I truly admire. Thank you for helping me improve as a researcher.

I also want to thank Prof. Dr.-Ing. Martin Theobald for his support, motivation and for the effort he put on the papers. Thank you for giving me a refreshing research perspective.

I would also like to thank Prof. Dr. Abraham Bernstein, who agreed to be the third reviewer in my doctoral examination committee and Prof. Dr. Thomas Fritz for chairing the thesis defense.

I will always be grateful to Prof. Timos Sellis who guided me in the early stages of my research career and also encouraged me to apply for a doctoral position in the DBTG; his generosity to his students is rare.

A special thanks to all my colleagues and former colleagues from the DBTG at the University of Zurich. I appreciate their constructive feedback during our group meetings but I am mostly

thankful for our wonderful lunch discussions, and for their friendship.

A big thanks to all the friends who were patient enough to bear with me. Panos, thanks for the discussions and constant motivation.

Last but not least, there are four people without whom I wouldn't have made it through my PhD journey: my family. They believed in me, even in times when I didn't believe in myself and it's great to feel that they have my back.

Aikaterini Papaioannou
Zurich, February 2019

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1 Background

The need to manage large, temporal-probabilistic (TP) datasets appears in a wide range of applications. Data cleaning, for example, is treated as the problem of reducing uncertainty in data and can be facilitated by the existence of timestamps that strengthen or weaken the proximity between entities. Scientific databases, that store outcomes of scientific experiments, contain uncertain data, such as incomplete observations or imprecise measurements, with a timestamp that indicates the time point when a measurement was taken or the interval during which it is valid. RFID data can be used to track the position of moving objects at different time points and such measurements are associated with the system's confidence in each validity [WKL+08].

Example 1. *Consider Migros, the leading supermarket branch in Switzerland that has multiple stores all over the country and that also offers the possibility to order groceries via its website. The supermarket records data related to purchases of clients, online shopping carts, and inventory, i.e. information that can be used to predict the behavior of its clients and to guarantee they will not leave a store or cancel an order unsatisfied due to unavailability of an item they intended*

to buy. In Table 1.1, relation **A** *contains the products that customers buy during a month at Migros' stores in Zurich, relation* **B** *contains the products that customers order via the website and relation* **C** *contains the products that are in stock in each store. Abbreviations HB and OE refer to the stores in Hauptbahnhof (Central Station) and in Oerlikon, respectively. M-budget, Farmer, Alnatura and Milupa are four of the brands with products sold in Migros. More specifically, tuple* a_1 *from relation* **A** *indicates that, at each day from the* 1^{st} *until the* 8^{th} *of the month, 'Ann shops cola in the physical store in HB' with probability 0.5. There is no other tuple in* **A** *that predicts the probability of 'Ann buying cola' over an interval overlapping with* $[01,08)$.

Table 1.1: The Supermarket Application Scenario

A (buysAt)

Name	*Product*	*Location*	λ	T	p
Ann	cola	HB	a_1	[01,08)	0.5
Liz	muesli	HB	a_2	[05,16)	0.7
Bob	chips	OE	a_3	[15,21)	0.8

B (ordersFrom)

Name	*Product*	*Location*	λ	T	p
Ann	cola	HB	b_1	[04,08)	0.4
Cam	salad	OE	b_2	[20,27)	0.6
Nick	juice	OE	b_3	[02,13)	0.5

C (inStockIn)

Product	*Brand*	*Location*	λ	T	p
muesli	Farmer	HB	c_1	[07,11)	0.6
cola	M-budget	HB	c_2	[04,08)	0.6
muesli	Milupa	OE	c_3	[27,29)	0.8
muesli	Alnatura	HB	c_4	[09,17)	0.8

Assume the supermarket wants to determine, at each time point, the probability that a client intends to buy a product in a physical store and not online, and the product is in stock in this store. The algebra expression for this query is:

$$Q = (\mathbf{A} \; -^{Tp} \; \mathbf{B}) \; ⟕^{Tp} \; \mathbf{C}$$

i.e., the set-difference of relations **A** *and* **B***, followed by a left outer-join with relation* **C***. The result of the temporal-probabilistic set-difference includes the products that a client buys in a physical store but does not order from the website. The result of the temporal-probabilistic left*

outer-join computes at each time point, for all clients, the probability for each of their requested products to be in stock.

Time and probability have been studied for several years as two separate dimensions. Temporal works have focused on the establishment of proper semantics [BBJ98, Tom98], on the integration of temporal algebra in a DBMS [Jen00, B$\ddot{\text{O}}$5, DBGJ16a, BJS00, LM97] and on the efficient computation of output intervals [CB17, DBG12, KMV$^+$13]. On the other hand, probabilistic works have focused on modeling uncertain data [JOS10, BSH$^+$08, Koc08], on developing algorithms that compute output probabilities [STW08, FOR11, GS17] and on the properties of a query that simplify the required computations [OHK10, FHO13], when possible [FO16, DS12, GS14].

Inherited from temporal and probabilistic databases, computing algebra operations on interval-timestamped relations with probabilities lays in the manipulation of intervals and lineage expressions as well as in the computation of output probabilities. The works that have focused on the combination of the two dimensions are limited [DRS01, DMT13]. They have introduced data models and they have expressed temporal-probabilistic operations using SQL or datalog enhanced with temporal predicates. These approaches do not support TP operations with negation, namely TP set-difference, TP outer-joins and TP anti-join and suffer from redundant computations since they treat the two dimensions in a decoupled manner.

This thesis is about (i) an approach for the efficient computation of TP set operations based on recording candidate intervals together with the contributing input lineages and making on the spot decisions on the usability of the candidate for the output; (ii) an approach for the computation of TP outer-joins and TP anti-join that avoids redundant computations by using a computational basis and delaying the concatenation of input lineages until all output information is produced; and (iii) three visualization charts that provide insights in the result of a temporal-probabilistic query.

1.2 Challenges

The first challenge arising from the combination of the temporal and the probabilistic dimension lies in the need for the result of relational algebraic operators to comply with the semantics of each dimension. Probabilistic databases rely on the *possible-worlds semantics* to define for which instances of the probabilistic database an answer tuple is valid. Conversely, temporal databases use the *sequenced semantics* to define at which time points (i.e., snapshots of the

temporal database) an answer tuple is valid. As a consequence, we perceive query evaluation in a TP database as the evaluation of a query over all the possible instances of the database at each time point.

The reason why the possible-worlds and the sequenced semantics very nicely complement each other is that they both employ the notion of *data lineage* to guarantee a closed and complete representation model for temporal, uncertain data. In temporal databases, lineage is used in the form of a set, as a means to guarantee that the output intervals of a query are maximal. In other words, an interval is formed when a fact is valid over consecutive time points at which the fact has been derived based on the same input tuples. Outside the interval, different tuples contribute to the fact. In probabilistic databases, lineage is used in the form of a boolean expression. It serves as a concise condition that is satisfied over the possible worlds in which an answer tuple exists. These are the possible worlds whose probabilities are eventually summed for the computation of the probability of the answer tuple.

In this work, we introduce a TP model where lineage (λ) is used in the form of a boolean expression and it has a twofold role. It serves both for the guarantee of merging the result of consecutive time points into maximal intervals but also as a means of combining the probabilities of possible worlds that yield the same result at each of the time points of an interval. As an immediate consequence of a combined model, we explore how the computation of algebra operations on TP relations benefit from the connection between lineage and intervals, with emphasis on binary operations with negation.

Table 1.2: The result of $Q = (\mathbf{A} \ -^{\text{Tp}} \ \mathbf{B}) \bowtie^{\text{Tp}} \mathbf{C}$

Name	*Product*	*Location*	*Brand*	λ	T	p
Ann	cola	-	-	a_1	[01,04)	0.5
Ann	cola	HB	M-budget	$(a_1 \wedge \neg b_1) \wedge c_2$	[04,08)	0.180
Liz	muesli	HB	-	$a_2 \wedge \neg c_1$	[05,07)	0.7
Liz	muesli	HB	Farmer	$a_2 \wedge c_1$	[07,11)	0.42
Liz	muesli	HB	-	$a_2 \wedge \neg c_1$	[07,09)	0.28
Liz	muesli	HB	Alnatura	$a_2 \wedge c_4$	[09,16)	0.56
Liz	muesli	HB	-	$a_2 \wedge \neg(c_1 \vee c_4)$	[09,11)	0.056
Liz	muesli	HB	-	$a_2 \wedge \neg c_4$	[11,16)	0.14
Bob	chips	OE	-	a_3	[15,21)	0.8
Ann	cola	HB	-	$(a_1 \wedge \neg b_1) \wedge \neg c_2$	[04,08)	0.120

Example 2. *In Table 1.2 we present the result of the query* $Q = (\mathbf{A} \ -^{Tp} \ \mathbf{B}) \bowtie^{Tp} \mathbf{C}$ *where* **A**, **B**, **C** *are the relations in Table 1.1. The output tuple 'Ann, cola, HB, M-budget',* $(a_1 \wedge \neg b_1) \wedge c_2$,

[04,08), 0.180) indicates that for each time point from the 4^{th} until the 8^{th} of the month, 'Ann buys cola from the brand M-budget in the store in HB and cola is in stock' with probability 0.180. At each time point in $[04,08)$*, for the fact ('Ann,cola, HB) to be true, the following must hold in a possible state of the database: (a) Ann intends to buy cola from M-budget in HB (a_1), (b) Ann does not order cola online from the store in HB ($\neg b_1$), and (c) Cola from M-Budget is available in HB (c_1). These conditions are reflected in the lineage expression (λ) of this fact that is $(a_1 \wedge \neg b_1) \wedge c_2$. Given the lineage expression, the probability can be computed by multiplying the input probabilities $a_1.p$ (0.5), $1-b1.p$ (0.6) and $c_1.p$ (0.6). The interval* $[04,08)$ *is maximal, i.e., it cannot be shortened or enlarged. It cannot be shortened because if it did, we would exclude time points that have the same lineage expression as the time points included. Similarly, it cannot be enlarged because at time points outside* $[04,08)$*, there is a change in the input tuples that contribute to the fact '(Ann, cola, HB, M-budget)' since tuple b_1 is no longer valid. The output tuple ('Liz, muesli, HB, -', $a_2 \wedge \neg c_1$, [07,09), 0.28) indicates that for each time point from the 7^{th} until the 9^{th} of the month, 'Liz wants to buy muesli in HB' and muesli from any brand is not in stock with probability 0.28.*

Operations with negation are performed over a positive relation **p** and a negative relation relation **n**, given a θ condition that determines the tuples that match. In conventional databases, operations with negation disqualify an input tuple of the positive relation if its non-temporal attributes match the attributes in a tuple of the negative relation according to θ. In temporal databases, the existence of a matching tuple in the negative relation does not disqualify the tuple of **p** itself but time points at which it is valid [BBJ98, BJ09]. In probabilistic databases, where tuples have a probability to be true or false, the existence of a matching tuple in **n** only reduces the probability with which a tuple is included in the output [Suc09, WRS08].

The result of a temporal-probabilistic operation with negation includes, at each time point, the probability with which a tuple of the positive relation **p** matches no tuple in the negative relation **n** under a predicate θ. Firstly, it includes output tuples that span subintervals when only tuples of the positive relation **p** are valid. In such cases, output intervals might be determined by starting or ending points of input tuples that are not valid during the output interval. Secondly, TP operations with negation produce outputs that indicate, at each time point, the probability of a tuple p_1 in the positive relation not matching any valid tuple in the negative relation because all of them are false. In this case, an output interval T is determined based on the starting and ending points of p_1 and of all the tuples of **n** that are valid over T and match p_1 under θ.

TP set-difference is a TP operation with negation in which the θ condition requires that the combined input tuples to have pairwise equal non-temporal attributes. Moreover, given the assumption of the TP model that only one tuple of each relation can include a fact at a time point and given that only tuples with equal facts are combined, a tuple of the left relation can be combined with at most one tuple of the right relation. These two conditions hold for all TP set operations and they guarantee, similarly to temporal and probabilistic databases, that set queries in temporal-probabilistic databases yield linearly sized output relations. However, all of the existing solutions exhibit a quadratic runtime complexity.

Despite the overhead added in their computation when combining the temporal and probabilistic dimension, TP set operations have received little attention so far: they have not been explicitly defined in existing TP approaches [DMT13], with TP set difference not being supported at all. When a probabilistic dimension exists, set operations are reduced to joins [DMT13, FOR11], since their computation not only requires the comparison of relational attributes among the input tuples, but also the combination of their lineage expressions. Moreover, the computation of output intervals under a sequenced TP data model requires more sophisticated solutions than the use of temporal predicates in joins and, for this purpose, existing temporal techniques [DBGJ16b, DBG12] rely on quadratic joins with inequality conditions.

In this thesis, we improve over the complexity of TP set operations based on two observations. Firstly, the assumption that at most one tuple of each intput relation influences the result at each time point allows for the use of a more efficient technique than inequality joins for the identification of output intervals. Secondly, the interdependence of lineage and intervals under a sequenced TP model is the key to overcoming their decoupled computation and thus we use lineage as a means to decide on whether an output tuple for a given TP operation can be created at a time point.

In the case of TP outer-joins and TP anti-join, the θ condition that is applied can be more general than strict equality on all non-temporal attributes, used in TP set operations. As a result, multiple tuples of the negative relation might be valid over an output interval and input tuples with different non-temporal attributes might be combined to form an output tuple. On top of that, TP outer-joins combine the characteristics of TP operations with and without negation: at each time point, two outcomes might arise based on the same valid tuples since they can be either $true$ or $false$. For example, in the result of the TP outer-join in Fig. 1.2, tuple ('Liz,muesli,HB,Farmer', $a_2 \wedge c_1$, [07,11), 0.42) indicates that for each time point in [07, 11), 'Liz wants to buy muesli in HB' (a_2) and 'muesli from the brand Farmer is available in HB' (c_1) with probability 0.42. This

tuple has been derived from input tuples a_2 and c_1 and its interval corresponds to the intersection of the intervals of these two input tuples. On the contrary, the interval of tuple ('Liz, muesli, HB, -', $a_2 \wedge \neg(c_1 \vee c_4)$, [09,11), 0.056) corresponds to the overlap of all three tuples included in its lineage.

In temporal-probabilistic approaches that only perform pairwise comparisons for the computation of joins, output tuples related with negation cannot be produced [DMT13]. Temporal approaches [DBGJ16b] require adjusting the input relations and recombining adjusted results to produce output facts and lineages. Adjusting input tuples and combining the adjusted results leads to redundant comparisons and to recomputation of intermediate results. Most importantly, similarly to TP set-operations, existing approaches suffer from the decoupled computation of output intervals, facts and lineages due to the overhead it adds in the computational time. The solution we have proposed is based on the following observation: the cases covered in the output of TP anti-join and outer-join can all be computed based on the pairs of tuples that overlap at a time point. As a result, the computation of this basis is only computed once and the output is produced in an incremental manner.

The question posed after the result of a TP query has been computed is whether it is insightful enough in the analysis of the output data. Consider the case of the supermarket and more specificaly the output tuple u = ('Liz, muesli, HB, -', $a_2 \wedge \neg(c_1 \vee c_4)$, [09,11), 0.056) in Fig.1.2. This tuple indicates that 'Liz wants to buy muesli in HB' and muesli from any brand is not in stock, with probability 0.056. If Migros wants to decrease this probability, so that it is less possible that Liz visits the store in HB without finding a product matching her preferences. Given such a goal, the information in the output is clearly insufficient. It is not possible to precisely estimate the probability of u if the probability that Liz wants to buy muesli in HB" is increased. Similarly, it is unclear whether a larger decrease is achieved by modifying the probability that "muesli from brand Farmer is in stock" or by modifying the probability that "muesli from brand Alnatura is in stock".

We resolve the issues raised in this example in **TemProRA**, a system focusing on the analysis and tuning of the k result tuples of a temporal-probabilistic query with either the lowest or the highest probabilities. For each output tuple u, we propose three key visualization tools to illustrate: (i) the input tuples contributing to the creation of u (TP Lineage Tree), (ii) the impact that the probability of each input tuple in the lineage of u has on the probability of u (TP Bubble Chart), and (iii) the new value for the probability of u when an input probability is modified (TP Column Chart).

1.3 Contributions

This thesis makes three main contributions to the database field:

- It simplifies the computation of TP set operations by introducing lineage-aware temporal windows, a mechanism that binds the output intervals with the lineage expressions of the pair of valid tuples, thus allowing for the filtering and finalization of candidate output tuples at the time when they are produced.
- It introduces the generalized lineage-aware temporal windows for the computation of TP outer-joins and TP anti-join where multiple tuples of each input relation can be valid at a time point. The windows are grouped into three disjoint categories that can be computed incrementally, allowing for a major improvement in the runtime of TP joins with negation.
- It introduces three types of charts for the analysis of the top-k temporal-probabilistic results of a query, providing the user with the most important information to systematically modify the time-varying probability of result tuples.

The research methodology that has been adopted for each part of this thesis starts with a problem given from real world followed by an analysis and precise definition of the problem. The solution to a problem and its properties are studied, elaborated analytically and then implemented. Large parts of this thesis have been implemented into the open source database system PostgreSQL and made available as open source (https://www.ifi.uzh.ch/en/dbtg/research). The implementation is extensively evaluated and compared with state-of-the-art approaches to confirm the analytical results of the solution. The rest of this section elaborates the contributions of this thesis in more detail with examples.

1.3.1 TP Set-Operations

The first contribution of this thesis is the definition and computation of TP set operations under a sequenced temporal-probabilistic data model that we introduced. The query semantics of the model comply with both the sequenced semantics from temporal databases [BJ09] and the possible-worlds semantics from probabilistic databases [SOCK11]. As a result, conceptually, query evaluation resolves to evaluating a query at every possible state the database can be at each time point. In practice, the query semantics of our sequenced TP data model are based on data

lineage, the bridge between the need to create maximal intervals [temporal dimension] and to compute probabilities based on boolean expressions [probabilistic dimension].

A (buysAt)

Name	*Product*	*Location*	λ	T	p
Ann	cola	HB	a_1	[01,08)	0.3
Liz	muesli	HB	a_2	[05,16)	0.2
Bob	chips	OE	a_3	[15,21)	0.4

B (ordersFrom)

Name	*Product*	*Location*	λ	T	p
Ann	cola	HB	b_1	[04,08)	0.4
Cam	salad	OE	b_2	[20,27)	0.6
Nick	juice	OE	b_3	[02,13)	0.5

(a) Input Relations

	F = '*Name*, *Product*, *Location*'	λ_r	λ_s	T
w_1	'Ann, cola, HB'	a_1	`null`	[01,04)
w_2	'Nick, juice, OE'	`null`	b_3	[02,13)
w_3	'Ann, cola, HB'	a_1	b_1	[04,08)
w_4	'Liz, muesli, HB'	a_2	`null`	[05,16)
w_5	'Bob, chips, OE'	a_3	`null`	[15,21)

(b) The lineage-aware temporal windows of **A** and **B**

Figure 1.1: The Lineage-Aware Temporal windows of relations **A** and **B**

After formally defining TP set operations and studying the properties of TP set queries, we introduce the lineage-aware temporal window as a mechanism that associates candidate output intervals with the lineage expressions of the valid input tuples. The lineage-aware temporal window is built for set operations and it has schema (F, `winTs`, `winTe`, $\lambda_{\mathtt{r}}$, $\lambda_{\mathtt{s}}$). F is a fact included in tuples over the interval $[\mathtt{winTs}, \mathtt{winTe})$. $\lambda_{\mathtt{r}}$ and $\lambda_{\mathtt{s}}$ are the lineage expressions of the input tuples of the left input relation **r** and the right input relation **s**, respectively, that are valid over $[\mathtt{winTs}, \mathtt{winTe})$ and include F. For example, according to the window w_3 in Tab. 1.1, at each time point in [04,08), the fact 'Ann, cola, HB' is included in tuple a_1 of relation **A** and in tuple b_1 of relation **B**. According to w_1 and w_2, respectively, the fact 'Ann, cola, HB' is included in no tuple of relation **B** during [01,04) and the fact 'Nick, juice, OE' is included in no tuple of relation **A** during [02,13).

The lineage-aware window-advancer (LAWA) is the algorithm we introduce for the creation of lineage-aware temporal windows. LAWA produces lineage-aware temporal windows whose left (`winTs`) and right (`winTe`) boundaries are computed during a sweep of the start (`Ts`) and end (`Te`) points of the tuples of two duplicate-free TP relations with schema (F, λ, T, p). LAWA's simplicity derives from the main assumption in TP databases that affects the result of TP set operations: only one tuple of an input relation describes the probability of a fact being true at a time point. Thus, its right boundary $\texttt{winTe}_i$ is the smallest among four time points at most: (a) the end points of the two tuples expected to overlap with this window, i.e., tuples with $\texttt{Ts} \leq \texttt{winTs}$ and $\texttt{Te} > \texttt{winTs}$, and (b) the start points of the two tuples to be processed next. Given that the input relations are swept after having been sorted, LAWA guarantees linearithmic complexity for the computation of TP set operations, thus improving over the quadratic complexity of existing approaches that compute TP set operations.

Name	*Product*	*Location*	λ	T	p
Ann	cola	HB	a_1	[01,04)	0.5
Ann	cola	HB	$a_1 \wedge \neg b_1$	[04,08)	0.20
Liz	muesli	HB	a_2	[05,16)	0.7
Bob	chips	OE	a_3	[15,21)	0.8

Figure 1.2: The result $\mathbf{U} = \mathbf{A}\ -^{\text{Tp}}\ \mathbf{B}$

The flexibility of the *lineage-aware temporal windows* is based on the fact that the lineages of valid tuples of each input relation are separately recorded. Exploiting the flexibility of a *lineage-aware temporal window*, we reduce the implementation of TP set operations into two additional steps. When a window is created, a lineage-based filter (λ_{filter}), different for each TP set operation, is directly applied. In contrast to previous works of either temporal or probabilistic set operations, this step involves no application of additional algebraic operations, no tuple replication and no redundant interval comparisons. After the filtering step, the window is finalized to an output tuple by applying the lineage-concatenating function ($\lambda_{function}$) of the respective TP set operation on $\lambda_{\texttt{r}}$ and $\lambda_{\texttt{s}}$. For example, in the case of TP set-difference, the filter applied is $\lambda_r \neq \texttt{null}$, i.e. it requires for a tuple of the left relation to be valid. Thus, for the computation of $\mathbf{A}\ -^{\text{Tp}}\ \mathbf{B}$, on the relations of **A** and **B** of Tab. 1.1, the window w_3 in Tab. **??** is transformed to the output tuple ('Ann, cola, HB', $a_1 \wedge \neg b_1$, [04,08), 0.20) (Tab. 1.2). On the contrary, window w_2 is filtered out and leads to no output tuple. The computation of the output probabilities can be performed using established either exact (see, e.g., [DS07, DS12, OH08]) or approximate (see, e.g., [FHO13, FO11, GS14, GS15, OHK10]) methods.

1.3.2 TP Outer-Joins and Anti-Join

The second contribution of this thesis is the introduction and use of generalized lineage-aware temporal windows for the computation of TP outer-joins and TP anti-join. Similarly to lineage-aware temporal windows, they bind output intervals with the lineage expressions of the tuples valid over this interval. However, they are able to cater for two additional requirements of joins with negation: the pairing of input tuples that include different facts and also the combination of multiple input tuples that are valid over an interval and satisfy θ. Given these requirements and the benefits from the simultaneous computation of output intervals and lineages, we introduce ***generalized lineage-aware temporal windows*** with schema (F_r, F_s, T, λ_r, λ_s). F_r is a fact included in tuples of relation **s** over the interval T while F_s is a fact included in tuples of relation **s** over the interval T. λ_r is the disjunction of the lineage expressions of the input tuples of the left input relation **r** that are valid over T and include F_r and satisfy a given θ-condition. λ_s is the disjunction of the lineage expressions of the input tuples of the right input relation **s** that are valid over T and satisfy a given θ-condition.

The ***generalized lineage-aware temporal windows*** produced with two TP relations and a θ-condition are grouped into three disjoint sets based on the case of TP joins with and without negation that they cover. The *unmatched windows* $\mathbf{W}_U(\mathbf{r};\mathbf{s},\theta)$ correspond to output tuples that cover the case of a tuple of relation **r** not matching a tuple of relation **s**. The overlapping windows $\mathbf{W}_o(\mathbf{r};\mathbf{s},\theta)$ cover the case of output tuples for TP joins without negation. The set of negating windows $\mathbf{W}_N(\mathbf{r};\mathbf{s},\theta)$ of the TP relation **r** with respect to the TP relation **s** includes lineage-aware temporal windows during which a fact is included in a tuple r of **r** as well as in multiple tuples of **s** that are valid and satisfy the θ-condition.

Example 3. In Tab. 1.3, we illustrate all sets of generalized lineage-aware temporal windows that are produced from relation **U** with respect to relation **C** using the θ-condition: $U.Location = C.Location \wedge U.Product = C.Product$. According to the overlapping window w_{o1}, at each time point in [04,08), the fact 'Ann, cola, HB' included in a tuple of **U** with lineage $a_1 \wedge \neg b_1$ is combined with the fact 'cola, M-budget, HB' in the tuple of **C** with lineage c_2 of relation. According to the window w_{n3}, during [09,11), the fact 'Liz, muesli, HB' matches no tuple in **C** ($F_s = \texttt{null}$) if all tuples matching the θ-condition are false, i.e. if the disjunction of their lineages $c_2 \vee c_4$ is false.

The main characteristic of our approach is that we record the lineages of the tuples valid in the left and right relation over an output interval and we keep them decoupled, in the form

	F_{L}	F_{R}	T	λ_{L}	λ_{R}
w_{u1}	'Ann, cola, HB'	null	[01,04)	a_1	null
w_{u2}	'Bob, chips, OE'	null	[15,21)	a_3	null
w_{u3}	'Liz, muesli, HB'	null	[05,07)	a_2	null

(a) Unmatched windows: $\text{W}_u(\mathbf{U};\mathbf{C},\theta)$

	F_{L}	F_{R}	T	λ_{L}	λ_{R}
w_{o1}	'Ann, cola, HB'	'cola, M-budget, HB'	[04, 08)	$a_1 \wedge \neg b_1$	c_2
w_{o2}	'Liz, muesli, HB'	'muesli, Farmer, HB'	[07,11)	a_2	c_1
w_{o3}	'Liz, muesli, HB'	'muesli, Alnatura, HB'	[09,16)	a_2	c_4

(b) Overlapping windows: $\text{W}_o(\mathbf{U};\mathbf{C},\theta)$

	F_{L}	F_{R}	T	λ_{L}	λ_{R}
w_{n1}	'Ann, cola, HB'	null	[04, 08)	$a_1 \wedge \neg b_1$	c_2
w_{n2}	'Liz, muesli, HB'	null	[07,09)	a_2	c_1
w_{n3}	'Liz, muesli, HB'	null	[09,11)	a_2	$c_1 \vee c_4$
w_{n4}	'Liz, muesli, HB'	null	[11,16)	a_2	c_4

(c) Negating windows: $\text{W}_n(\mathbf{U};\mathbf{C},\theta)$

Figure 1.3: The sets of generalized lineage-aware temporal windows of **U** with respect to **C**

of a generalized lineage-aware temporal window until all the windows required to produce the output of a join are formed. For the computation of all windows in the sets of unmatched and negating windows, we introduce the algorithms LAWA_U and LAWA_N, respectively. The set of overlapping windows is the basis for the computation of unmatched and negating windows for two TP relations **r** and **s**. In order to produce the unmatched windows of **r** with respect to **s**, we use LAWA_U to identify subintervals of **r** during which there is no overlap or match with a tuple of **s**, i.e. subintervals that do not correspond to any overlapping window. Similarly, each of the negating windows of **r** with respect to **s** is produced using LAWA_N and spans a subinterval where all the tuples of **s** that overlap and match with a tuple r of **r** are false and thus lineage information from all the overlapping windows that are valid over this subinterval and involving r must be combined. For example, the intervals of windows w_{n2} and w_{n3} in Tab. 1.3 correspond to subintervals of [07,11) or [09,16), i.e. the intervals of the overlapping windows w_{o2} and w_{o3}, respectively. Given the decoupled lineages in the generalized lineage-aware temporal windows, we have the possibility to combine the information of valid tuples over a time point into multiple

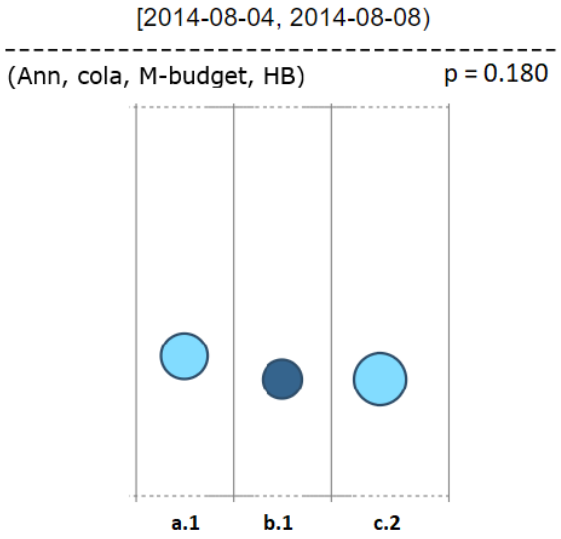
output tuples and thus avoiding redundant interval comparisons due to the repetition of basic steps. This leads to an improvement in the runtime required for the computation of outer-joins and anti-join by two orders of magnitude.

1.3.3 TP Result Analysis

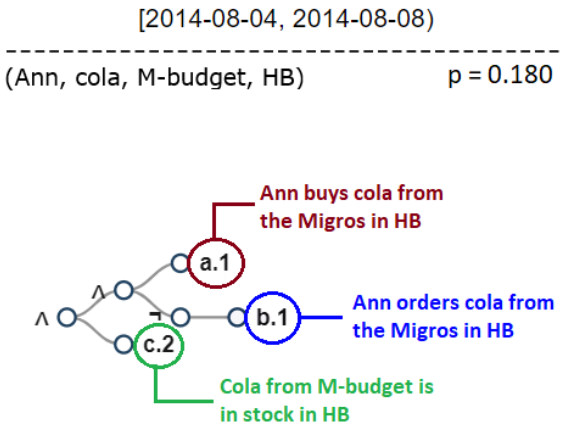
The third contribution of this thesis is three vizualization tools that provide the user with the most important information to systematically modify the time-varying probability of the top-k result tuples of a TP query. All three tools are based on the observation that the probability of an output tuple is linearly dependent on the probability of each of the input tuples in its lineage expression. The values of these coefficients of this linear dependency are computed and depicted so that a measure is developped for the influence of each input probability in the probability of an output tuple.

Thus, for each output tuple, we introduce the Temporal Probabilistic Lineage Tree (TPLT), the Temporal-Probabilistic Bubble Chart (TPBC) and the Temporal-Probabilistic Column Chart (TPCC). The Temporal-Probabilistic Bubble Chart illustrates the impact that a change in the probability of base tuples has on the probability of s. It focuses on the tuples with the highest impact and encodes whether it is positive or negative as well as its magnitude. The Temporal-Probabilistic Column Chart reports how a change (increase/decrease) in the probability of an output tuple is achieved via the modification of the probability of its most influential tuples. The Temporal-Probabilistic Lineage Tree visualizes the lineage of an output tuple, allowing the identification of all tuples influencing its probability by showing, for each point in time, how and from which base tuples this output tuple has been derived.

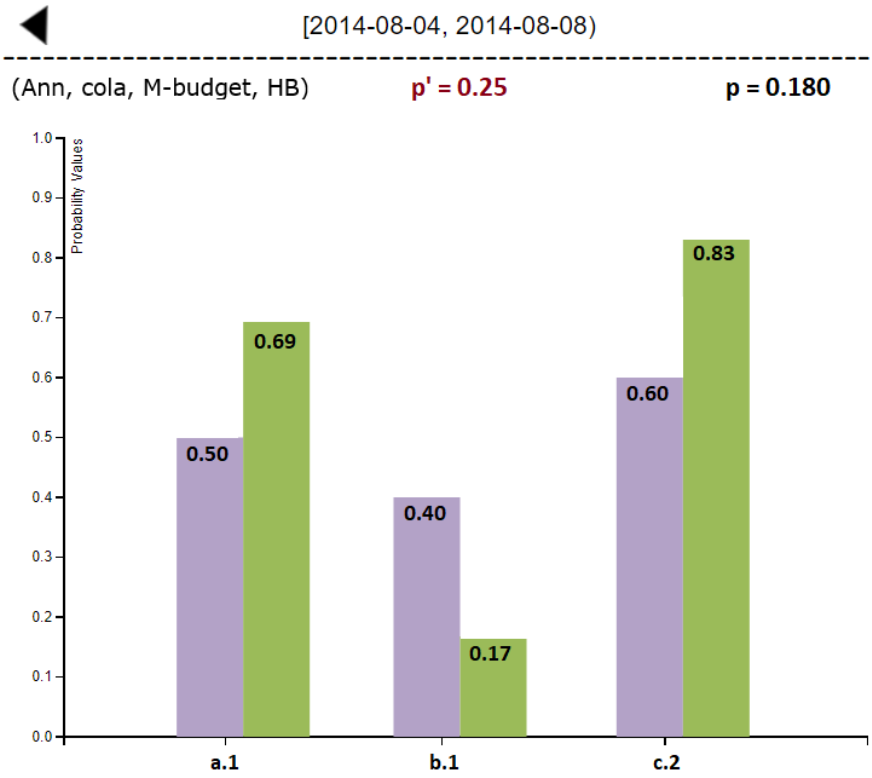
Example 4. In Fig. 1.4, we present the features of TemProRA for the output tuple q = (Ann, cola, M-Budget, HB, $(a_1 \wedge \neg b_1) \wedge c_1$, [04, 08), 0.180). This tuple indicates that, at each time point in [04, 08), there is 0.180 probability that 'Ann wants to shop cola in HB' and 'cola is in stock from the brand M-Budget in HB'. The Temporal-Probabilistic Bubble Chart (Fig. 1.4a) of q uses the color of the bubbles and their positioning in the y-axis to illustrate the probability impact of a_1, b_1, c_2 on $q.p$, i.e. how and how much a change in the probability of an input tuple will affect output probability. The bubble for a_1 is positioned higher than the other ones, with the probability impact valued $u_{a_1} = 0.360$, indicating that an increase of $b_1.p$ would lead to a greater increase in $q.p$ than the increase of $a_1.p, a_3.p, a_4.p$. Thus, the probability of tuple q is mostly influenced by the probability with which *Ann wants to buy cola* (a_1).



(a) TPBC



(b) TPLT



(c) TPCC

Figure 1.4: The Features of TemProRA on Tuple q_{13}.

If the user provides a new probability value $q.p' = 0.25$ for q, the TPCC (Fig. 1.4c) illustrates alternatives, involving the tuples a_1, b_1, c_2, in order to achieve this increase. One alternative is to increase $a_1.p$ by 0.19 (from 0.50 to 0.69) whereas another involves decreasing $b_1.p$ from 0.40 to 0.17. In case the user needs a more detailed view of why the tuples a_1, b_1, c_2 have been studied under the goal of increasing $q.p$, the TPLT of this tuple is provided in Figure 1.4b, illustrating the exact form of the lineage expression in a tree form.

1.4 Organization of the Thesis

This thesis is based on a collection of papers. A bibliography for all chapters is given at the end of the thesis.

Chapter 2 Lineage-Aware Temporal Windows for Temporal-Probabilistic Set Operations
K. Papaioannou, M. Theobald, and M. Böhlen, "Lineage-Aware Temporal Windows for Temporal-Probabilistic Set Operations" [invited submission for TKDE/ICDE 2018 Special Issue, including best-paper award candidates of ICDE 2018]

based on the paper: *"Supporting Set Operations in Temporal-Probabilistic Databases", K. Papaioannou, M. Theobald, and M. Böhlen, 2018 IEEE 34nd International Conference on Data Engineering (ICDE), Paris 2018*

Chapter 3 Outer-Joins and Anti-Join in Temporal-Probabilistic Databases
K. Papaioannou, M. Böhlen, and M. Theobald, "Outer-Joins and Anti-Join in Temporal-Probabilistic Databases," [Ready for submission]

Chapter 4 TemProRA: Top-k Temporal-Probabilistic Results Analysis
K. Papaioannou and M. Böhlen, "TemProRA: Top-k temporal-probabilistic results analysis," 2016 IEEE 32nd International Conference on Data Engineering (ICDE), Helsinki, 2016, pp. 1382-1385.
DOI: 10.1109/ICDE.2016.7498350

CHAPTER 2

Lineage-Aware Temporal Windows for Temporal-Probabilistic Set Operations

Abstract

In temporal-probabilistic (TP) databases, the combination of the temporal and the probabilistic dimension adds significant overhead to the computation of set operations. Although set queries are guaranteed to yield linearly sized output relations, existing solutions exhibit quadratic runtime complexity. They suffer from redundant interval comparisons and additional joins for the formation of lineage expressions. In this paper, we formally define the semantics of set operations in TP databases and study their properties. For their efficient computation, we introduce the *lineage-aware temporal window*, a mechanism that directly binds intervals with lineage expressions. We suggest the *lineage-aware window advancer* (LAWA) for producing the windows of two TP relations in linearithmic time, and we implement all TP set operations based on LAWA. By exploiting the flexibility of lineage-aware temporal windows, we perform direct filtering of irrelevant intervals and finalization of output lineage expressions and thus guarantee that no additional computational cost or buffer space is needed. A series of experiments over both synthetic

and real-world datasets show that (a) our approach has predictable performance, depending only on the input size and not on the number of time intervals per fact or their overlap, and that (b) it outperforms state-of-the-art approaches in both temporal and probabilistic databases.

2.1 Introduction

The need to manage large, temporal-probabilistic (TP) datasets appears in a wide range of applications, such as temporal predictions (e.g., weather) as well as in sensor (e.g., RFID) and other forms of scientific data, which are inherently temporal and frequently contain erroneous measurements. The combination of the temporal and the probabilistic dimension in a relational database setting requires that the result of the relational algebraic operators complies with the semantics of each dimension. To this end, probabilistic databases rely on the *possible-worlds semantics* to define for which instances of the probabilistic database an answer tuple is valid. Conversely, temporal databases use the *sequenced semantics* to define at which time points (i.e., snapshots of the temporal database) an answer tuple is valid. The possible-worlds and the sequenced semantics very nicely complement each other, since they both employ the notion of *data lineage* to guarantee a closed and complete representation model for temporal, uncertain data.

In this paper, we introduce a *sequenced* TP data model and, under this model, we define and implement the three principle TP set operations, *intersection* ($\cap^{Tp}$), *union* ($\cup^{Tp}$) and *difference* ($-^{Tp}$)[1]. In the following example, we illustrate the usefulness of TP set operators in an application involving temporal-probabilistic predictions.

Example 5. *Consider the supermarket application of Figure 2.1. The supermarket records data related to purchases of clients (***a***), online shopping carts (***b***), and inventory (***c***). At each time point (e.g., a day), the supermarket aims at predicting the products that clients want to buy or order versus those that it has in stock. For example, the tuple* ('milk', a_1, [2,10), 0.3) *captures that, at each day from the* 2^{nd} *to the* 10^{th} *of the month, "milk is bought" with probability 0.3. There is a single prediction for each fact at each time point and thus, there is no other tuple in* **a** *that predicts the probability of buying 'milk' over an interval overlapping with* $[2, 10)$.

In order to have an overview of its supply and demand, the supermarket wants to determine, at each time point, the probability that a product is in stock but no client wants to order or buy

[1]Note that, although in a relational setting *intersection* is a dependent operation which can be expressed in terms of two *difference* operations, we show that considering *intersection* as a separate operator has significant performance advantages in a TP setting.

a (productsBought)

Product	λ	T	p
'milk'	a_1	[2,10)	0.3
'chips'	a_2	[4,7)	0.8
'dates'	a_3	[1,3)	0.6

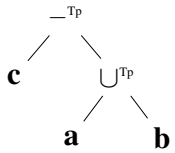
b (productsOrdered)

Product	λ	T	p
'milk'	b_1	[5,9)	0.6
'chips'	b_2	[3,6)	0.9

c (productsInStock)

Product	λ	T	p
'milk'	c_1	[1,4)	0.6
'milk'	c_2	[6,8)	0.7
'chips'	c_3	[4,5)	0.7
'chips'	c_4	[7,9)	0.8

(a) Input Relations



(b) Query

Product	λ	T	p
'milk'	c_1	[1,2)	0.6
'milk'	$c_1 \wedge \neg a_1$	[2,4)	0.42
'milk'	$c_2 \wedge \neg(a_1 \vee b_1)$	[6,8)	0.196
'chips'	$c_3 \wedge \neg(a_2 \vee b_2)$	[4,5)	0.014
'chips'	c_4	[7,9)	0.8

(c) Query Result

F	λ_r	λ_s	T
'milk'	a_1	null	[2,5)
'milk'	a_1	b_1	[5,9)
'milk'	a_1	null	[9,10)
'chips'	null	b_2	[3,4)
'chips'	a_2	b_2	[4,6)
'chips'	a_2	null	[6,7)
'dates'	a_3	null	[1,3)

(d) $\mathbf{W}(\mathbf{a},\mathbf{b})$

F	λ_r	λ_s	T
'milk'	c_1	null	[1,2)
'milk'	c_1	a_1	[2,4)
'milk'	null	a_1	[4,5)
'milk'	null	$a_1 \vee b_1$	[5,6)
'milk'	c_2	$a_1 \vee b_1$	[6,8)
'milk'	null	$a_1 \vee b_1$	[8,9)
'chips'	c_3	$a_2 \vee b_2$	[4,5)
'milk'	null	$a_2 \vee b_2$	[5,6)
'chips'	null	a_2	[6,7)
'chips'	c_4	null	[7,9)
'dates'	a_3	null	[1,3)

(e) $\mathbf{W}(\mathbf{c},\mathbf{a} \cup^{Tp} \mathbf{b})$

Figure 2.1: The Supermarket Application Scenario

this product. The corresponding query is $Q = \mathbf{c} -^{Tp} (\mathbf{a} \cup^{Tp} \mathbf{b})$*, i.e., the union of relations* **a** *and* **b***, followed by a difference with relation* **c** *(see Fig. 2.1b). Answer tuple* ('milk', $c_1 \wedge \neg a_1$, [2,4), 0.42) *(see Fig. 2.1c) expresses that, with probability 0.42,* 'milk' *is in stock but is not ordered or*

bought during interval $[2,4)$. *The lineage expression used for the computation of the interval and the probability of this tuple is formed based on the tuples of the input relations which are valid at each time point (*c_1 *and* a_1*) and the semantics of the operation to be computed (*$\cup^{Tp}$ *and* $-^{Tp}$*).*

TP set operations are interesting because of the overhead added in their computation when combining the temporal and probabilistic dimension. They are however a class of operations that have received little attention so far: they have not been explicitly defined in existing TP approaches [DMT13], with TP set difference not being supported at all. Existing temporal techniques suffer from two main drawbacks. First, approaches used for the computation of temporal set operations [DBGJ16b, DBG12] replicate input tuples with adjusted intervals before the actual algebraic operations are applied. They rely on joins with inequality conditions that have quadratic complexity due to unproductive tuple comparisons. Second, stitching lineage expressions to the output tuples in a relational manner requires additional joins in comparison to the set operations that are available in current temporal database implementations. Existing probabilistic approaches [FOR11], on the other hand, reduce set operations to joins, since their computation not only requires the comparison of relational attributes among the input tuples, but also the combination of their lineage expressions. However, the computation of TP set operations under a sequenced TP data model requires more sophisticated solutions for the computation of output intervals than the use of temporal predicates in joins.

In this paper, we introduce the concept of a *lineage-aware temporal window* as a means to combine the computation of the output intervals and the computation of the input lineage expressions that will contribute to an output tuple. The set of all windows of two TP relations constitutes a common core based on which we can produce the result of any TP set operation by using appropriate filter and concatenation functions. Based on this approach, we develop efficient algorithms for the computation of windows, and we eliminate redundancies in the steps that existing approaches need to rely on to identify the input tuples contributing to an output tuple.

Example 6. *In order to compute the query of Fig. 2.1b, we need to first compute the set of lineage-aware temporal windows* $\mathbf{W}(\mathbf{a},\mathbf{b})$ *of relations* **a** *and* **b** *(Fig. 2.1d) to compute their union. Each window spans a maximal interval over which a set of non-temporal attributes, called a "fact", is included in the same input tuples. The window w = ('milk',* a_1*,* b_1*, [5,9)) indicates that at each time point in [5,9), the fact 'milk' is included in the tuple of* **a** *with lineage* a_1 *and in the tuple of* **b** *with lineage* b_1.

In the result of a TP union, an output tuple is created when at least one of the input tuples is valid, and the windows of **a** *and* **b** *form output tuples by using a disjunction of the input lineages. Thus, window w is transformed into output tuple ('milk',* $a_1 \vee b_1$*, [5,9), 0.72). For the computation of the set difference* $\mathbf{c} -^{Tp} (\mathbf{a} \cup^{Tp} \mathbf{b})$*, the lineage-aware temporal windows of relations* **c** *and* $\mathbf{a} \cup^{Tp} \mathbf{b}$ *are computed as shown in Fig. 2.1e. The window ('milk',* c_2*,* $a_1 \vee b_1$*, [6,8)) indicates that at each time point in [6,8), the fact 'milk' is included in the tuple with lineage* c_2 *from input relation* **c***, while the tuple with lineage* $a_1 \vee b_1$ *is included from* $\mathbf{a} \cup^{Tp} \mathbf{b}$*, respectively. Note that the windows of Fig. 2.1e which are highlighted in red are not included in the final output of the TP set difference, since there is no valid tuple in the left input relation. An output tuple is created for each of the remaining lineage-aware temporal windows by concatenating the lineage expressions* λ_r *and* λ_s *to* $\lambda_r \wedge \neg\lambda_s$*.*

Outline & Contributions.

- We propose a *sequenced temporal-probabilistic data model* that complies with both the sequenced semantics from temporal databases [DBG12, BJ09] and the possible-worlds semantics from probabilistic databases [Suc09, SOCK11].
- We formally define the semantics of *TP set operations* and study the properties of TP set queries under this model. TP set queries have not previously been investigated under a sequenced temporal-probabilistic model.
- We introduce the concept of *lineage-aware temporal windows*, a mechanism that binds an interval with the lineages of the tuples that are valid during the interval. We show that each output tuple of a TP set operation maps to exactly one window, and we reduce the computation of a TP set operation between two TP relations to the application of conventional selection and projection operations over their sets of *lineage-aware temporal windows*.
- We introduce the *lineage-aware window advancer* (LAWA), a window-sweeping algorithm that computes all *lineage-aware temporal windows* of two TP relations and guarantees $O(n \log n)$ worst-case complexity. Exploiting the flexibility of the windows, we are able to finalize lineages and filter out irrelevant intervals directly at the time of their creation. No additional costs are involved and thus the computation of a TP set operation has linearithmic complexity, improving over existing implementations with quadratic complexity.
- We experimentally demonstrate that LAWA is the only approach that *does not deteriorate in performance* as the data history grows. In contrast to existing techniques, our solution

does not depend on the characteristics of the dataset (such as the number of intervals per fact, or the overlap among intervals), but only on the size of the input relations.

This paper is an extension of our ICDE paper [PTB18] and it is organized as follows. Section 3.2 provides an overview of related works on temporal and probabilistic databases with a focus on set operations. Section 2.3 introduces our TP data model, while Section 2.4 defines the model's query semantics. Section 2.5 defines TP set operations over duplicate-free input relations. Section2.6 introduces lineage-aware temporal windows. Section 2.7 introduces an algorithm for the computation of lineage-aware temporal windows, and Section 2.8 includes our implementation of TP set operations. Section 3.8 presents a comprehensive performance study that compares our implementation of TP set operations with existing timestamp-adjustment and lineage-computation approaches. Section 3.9 concludes the paper.

2.2 Related Work

We next review related approaches from both temporal and probabilistic databases and explain their limitations in terms of supporting TP set operations. Set difference, for example, has received little attention in temporal databases and can only be computed using the generic normalization operator [DBG12]. Under a combined temporal and probabilistic data model, there is currently no solution that supports set difference.

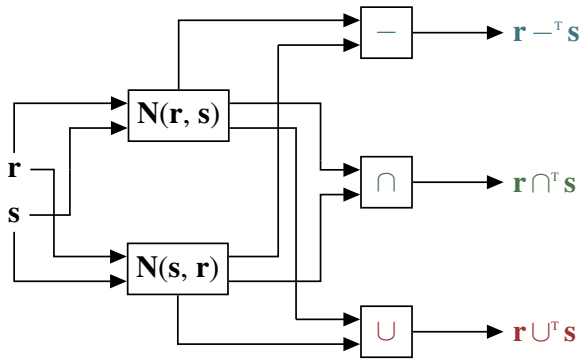


Figure 2.2: Temporal set operations using Normalize **N**.

Temporal Set Operations. In temporal databases, the result of a temporal set operation op^T is defined as the result of applying op over a sequence of atemporal instances (the so-called

snapshots) of the input relations—a key concept in temporal databases termed *snapshot reducibility* [AGC+13, LM97, VL07]. Maximal intervals are produced by merging consecutive time points to which the same input tuples have contributed (*change preservation*). Dignös et al. [DBG12, DBGJ16b] use *data lineage* to guarantee change preservation for all relational operations under a sequenced semantics. They adapt the *Normalization* operator, introduced by Toman et al. [Tom98], to compute temporal set queries. Intuitively, the normalization $N(\mathbf{r},\mathbf{s})$ of a relation $\mathbf{r}$ based on another relation $\mathbf{s}$ replicates the tuples of $\mathbf{r}$ and assigns new time intervals to them. The new intervals are obtained by splitting the original intervals based on tuples of $\mathbf{s}$ with which they overlap. Normalization is a generic operator that subsequently requires an outer join of $\mathbf{r}$ and $\mathbf{s}$ with quadratic complexity. Since it is not symmetric, it has to be computed once for each of the two input relations [DBG12, DBGJ16b] for the computation of temporal set-operations (cf. Fig. 2.2).

Temporal joins can be used for the computation of TP set intersection. Efficient solutions for temporal joins have been widely discussed in the literature [KMV+13, DBG14, PHD16, CB17]. Specific solutions either partition the data [CB17] in ways that are not beneficial for our case, since TP relations are duplicate-free (see Section 2.3), or they require fixed-length input schemas [PHD16]. *Timeline Index* (TI) is a data structure introduced by Kaufmann et al. [KMV+13, KVF+13] to efficiently compute temporal aggregation, join and time-travel operations. TI of relation $\mathbf{r}$ maps each start or end point in $\mathbf{r}$ to a list of ids of tuples that start or end at this time point. *Timeline Join* (TJ) is applied on the indexes created for the input relations and implements a combination of a merge- and a hash-join. The performance of TJ suffers because the original tuples need to be fetched both for the application of a filtering condition and for the creation of the output tuples.

Overlap Interval Partitioning (OIP) by Dignös et al. [DBG14] is designed to compute a join $\mathbf{r} \bowtie^T \mathbf{s}$ among tuples with overlapping time intervals. Initially, OIP splits the time domain into k granules of equal size. Adjacent granules are combined to form the partitions of an input relation $\mathbf{r}$ so that each tuple in $\mathbf{r}$ is assigned to the smallest partition into which it fits. In order to compute the overlap join, the overlapping partitions of $\mathbf{r}$ and $\mathbf{s}$ are identified (fast), and then a nested loop is performed to join the tuples of these partitions (slow). This approach finds all pairs of tuples (r, s), for $r \in \mathbf{r}$ and $s \in \mathbf{s}$, with overlapping time intervals. Although OIP can be extended to apply additional filtering conditions, e.g., equality conditions on the atemporal attributes of the tuples that are joined, its performance deteriorates when the condition has low selectivity (see Section 3.8).

Sweeping-based approaches, finally, have been widely used for the computation of overlap joins [PHD16, APR+98] in temporal settings. A sweepline moves over all start and end points of tuples, and determines, for each time point, the tuples of both input relations that are valid. These approaches cannot directly be applied for the computation of TP set operations. First, they generally do not consider join conditions on the non-temporal attributes. Second, they support set intersection but cannot produce all output tuples needed for set difference and union. The creation of output intervals through the tuples that the sweepline intersects is not sufficient for these two set operations.

Probabilistic Set Operations. In probabilistic databases, the result of a probabilistic set operation op^p is defined as the result of applying op over the set of all possible instances of the input relations. The Trio system [STW08] was among the first to recognize *data lineage*, in the form of a Boolean formula, as a means to capture the possible instances at which an output tuple is valid. In an effort to provide a *closed and complete* representation model for uncertain relational data, they introduced *Uncertainty and Lineage Databases* (ULDBs) [BSH+08]. The algebraic operators are modified to compute the lineage of the result tuples in a ULDB, thus capturing all information needed for computing query answers and their probabilities. Recently, Fink et al. [FOR11, FO16] reduced the computation of probabilistic algebraic operations to conventional operations (cf. Fig. 2.3) so that these can be performed using a DBMS, rather than by an application layer built on top of it.

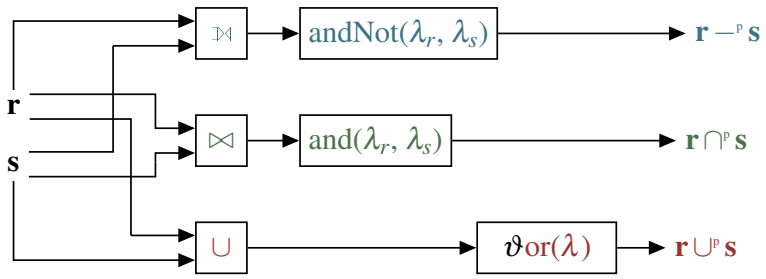


Figure 2.3: Probabilistic set operations. The joins filter out the facts that are not needed for the result and they add the input lineages in the same schema, so that output lineages can be formed using lineage-concatenating functions.

Temporal-Probabilistic Set Operations. A temporal-probabilistic model was introduced in the work of Dekhtyar et al. [DRS01]. Each tuple includes a TP part consisting of two temporal conditions, corresponding to sets of potential starting and ending points, and a pair of probability

values, corresponding to the minimum and the maximum probability of the fact being true. Conceptually, TP relations are converted into *annotated relations*, i.e., relations with tuples at a time-point granularity, and they are queried using annotated operators. The result is converted back to the initial compact representation, using probability combination functions. The use of these functions instead of lineage information has two implications. Firstly, change preservation [DBG12], a property of the temporal domain is not satisfied, since lineage is not used as a criteria to merge the results of consecutive time points into maximal intervals. Secondly, the closure property [IL84, SOCK11] of the probabilistic domain is not satisfied, since we lose track of the input tuples used for computing the probability of an output tuple, thus making the final result non-compositional.

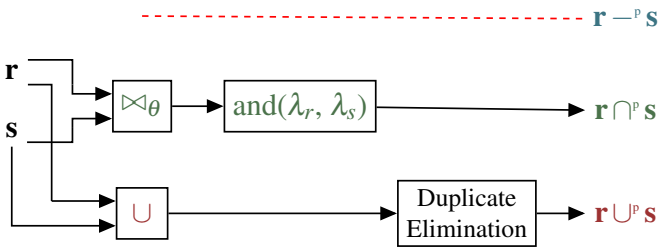


Figure 2.4: TP set operations in TPDB. Condition θ includes temporal predicates and duplicate elimination forms output intervals.

Dylla et al. [DMT13] introduced a closed and complete TP database model, coined TPDB, based on existing temporal and probabilistic concepts. Query processing is performed in two steps (cf. Fig. 2.4). The first step, grounding, evaluates a chosen deduction rule (formulated in Datalog with additional time variables and temporal predicates) and computes the lineage expressions of the deduced tuples. The second step, deduplication, removes the duplicates that could occur in the grounding step by adjusting their intervals. Although the TPDB data model is generic, the grounding step cannot cover operations whose results include subintervals that are only present in one of the two input relations. As explained in Section 2.5, sequenced TP set difference is one of these operations and is not supported by TPDB.

2.3 Data Model & Notation

We denote a **temporal-probabilistic schema** by $R^{\text{Tp}}(F, \lambda, T, p)$, where $F = (A_1, A_2, \ldots, A_m)$ is an ordered set of attributes, and each attribute A_i is assigned to a fixed domain Ω_i. λ is a

Boolean formula corresponding to a lineage expression. T is a *temporal attribute* with domain $\Omega^T \times \Omega^T$, where Ω^T is a finite and ordered set of *time points*. p is a *probabilistic attribute* with domain $\Omega^p = (0,1] \subset \mathbb{R}$. A **temporal-probabilistic relation r** over R^{Tp} is a finite set of tuples. Each tuple $r \in \mathbf{r}$ is an ordered set of values in the appropriate domains. The value of attribute A_i of r is denoted by $r.A_i$. The conventional attributes $F = (A_1, A_2, \ldots, A_m)$ of tuple r form a so-called *fact*, and we write $r.F$ to denote the fact f captured by tuple r. For example, the tuple ('milk', a_1, $[2,10)$, 0.3) of relation **a** (see Fig. 2.1a) includes the fact $a_1.F$ = ('milk'), the lineage expression $a_1.\lambda = a_1$, the time interval $a_1.T = [2,10)$, and the probability value $a_1.p = 0.3$. The temporal-probabilistic annotations of the schema express that (i) $a_1 = \mathit{true}$ with probability $a_1.p$ for every time point in $a_1.T$, (ii) $a_1 = \mathit{false}$ with probability $1 - a_1.p$ for every time point in $a_1.T$, (iii) and a_1 is always *false* outside $a_1.T$.

By following conventions from [DMT13, DBGJ16b, DBG12, OHK09], we assume duplicate-free input and output relations. Formally, a temporal-probabilistic relation **r** is **duplicate-free** iff $\forall r, r' \in \mathbf{r}(r \neq r' \Rightarrow r.F \neq r'.F \vee r.T \cap r'.T = \emptyset))$. In other words, the intervals of any two tuples of **r** with the same fact f do not overlap.

A **lineage expression** λ is a Boolean formula, consisting of tuple identifiers and the three Boolean connectives $\neg$ ("not"), $\wedge$ ("and") and $\vee$ ("or"). Tuple identifiers represent Boolean random variables among which we assume independence [DMT13, OHK09, DS07]). For a base tuple r, $r.\lambda$ is an atomic expression consisting of just r itself. For a result tuple $\tilde{r}$ derived from one or more TP operations, $\tilde{r}.\lambda$ is a Boolean expression as defined above. For a result tuple, lineage is determined by the temporal-probabilistic operators (formally defined in Section 2.4) that were applied to derive that tuple from the base tuples. The probability of a result tuple is computed via a probabilistic valuation of the tuple's lineage expression, using either exact (see, e.g., [DS07, DS12, OH08]) or approximate (see, e.g., [FHO13, FO11, GS14, GS15, OHK10]) algorithms. For example, in the result relation of Fig. 2.1c, the lineage $c_1 \wedge \neg a_1$ yields a marginal probability of $0.6 \cdot (1 - 0.3) = 0.42$ by assuming independence among the base tuples c_1 and a_1 (see Fig. 2.1a).

Finally, we write $\lambda_t^{\mathbf{r},f}$ as an abbreviation for:

$$\lambda_t^{\mathbf{r},f} = \begin{cases} r.\lambda & \text{iff } r \in \mathbf{r} \wedge r.F = f \wedge t \in r.T \\ \texttt{null} & \text{iff } \nexists\, r \in \mathbf{r}\,(r.F = f \wedge t \in r.T). \end{cases} \tag{2.1}$$

Thus, $\lambda_t^{\mathbf{r},f}$ refers to the lineage expression of a tuple in relation $\mathbf{r}$ with fact f that is valid at time point t. If there are no tuples in $\mathbf{r}$ with fact f at time point t, we write $\lambda_t^{\mathbf{r},f} = \texttt{null}$.

2.4 Query Semantics

For our query semantics, we adopt both the *sequenced semantics* [BJ09], widely used for the temporal dimension, and the *possible-worlds semantics* [SOCK11], commonly used for the probabilistic dimension. The sequenced semantics is consistent with viewing a temporal database as a sequence of atemporal databases (the "snapshots"), one for each time point t in Ω^T. Conceptually, query evaluation then resolves to evaluating a query against each of these snapshots and producing maximal output intervals according to time points with equivalent *data lineage*. Thus, an output interval consists of time points, in which the corresponding fact has been derived based on the same input tuples. The possible-worlds semantics defines a probabilistic database as a probability distribution over a finite set of possible states (aka. "worlds") in which the probabilistic database could be. Conceptually, a query is evaluated against each of the possible worlds. The marginal probability of an answer tuple then is defined as the sum of the possible-worlds probabilities, for which the answer tuple exists. Data lineage [BSH+08, STW08], in the form of a Boolean expression, serves as a concise condition that is satisfied over the possible worlds in which each answer tuple exists.

a (productsBought)

Product	λ	T	p
'milk'	a_1	[2,3)	0.3
'dates'	a_3	[2,3)	0.6

c (productsInStock)

Product	λ	T	p
'milk'	c_1	[2,3)	0.6

Figure 2.5: Probabilistic Snapshots $\tau_2^{\mathrm{p}}(\mathbf{a})$ and $\tau_2^{\mathrm{p}}(\mathbf{c})$

The query semantics of our sequenced TP data model is based on an intriguing analogy between the temporal and probabilistic semantics: rather than iterating over snapshots or possible worlds, they both use the notion of data lineage to define their operational semantics. Given a TP relation $\mathbf{r}$, a tuple $r \in \mathbf{r}$ is valid at every time point t included in its time interval $r.T$ with probability $r.p$. Thus, all tuples of a TP relation $\mathbf{r}$ that are valid at time point t with a given probability are included in the *probabilistic snapshot* of $\mathbf{r}$ at t. Specifically, we obtain the probabilistic snapshot of a TP relation $\mathbf{r}$ with schema $R^{\mathrm{Tp}} = (F, \lambda, T, p)$ at time point t by applying the *timeslice operator* τ_t^{p}, which is defined as:

$$\tau_t^{\mathrm{p}}(\mathbf{r}^{\mathrm{Tp}}) = \{(r.F, r.\lambda, [t, t+1), r.p) \,|\, r \in \mathbf{r} \wedge t \in r.T\}$$

In Fig. 2.5, we illustrate the probabilistic snapshots of the relations **a** and **c** of Fig. 2.1a at time point $t = 2$. The probabilistic snapshot of relation **b** at this time point is `null` since there is no tuple of **b** valid.

Definition 1. (TP Snapshot Reducibility) *Let $\mathbf{r}_1, \ldots, \mathbf{r}_m$ be a set of TP relations, let op^{Tp} be an m-ary temporal-probabilistic operator, let op^{p} be the corresponding probabilistic operator, let Ω^T be the time domain, and let $\tau_t^{p}(\mathbf{r})$ be the timeslice operator. The operator op^{Tp} is* snapshot reducible *to op^{p} iff, for all $t \in \Omega^T$, it holds that:*

$$\tau_t^{\mathrm{p}}(op^{\mathrm{Tp}}(\mathbf{r}_1, \ldots, \mathbf{r}_m)) \equiv op^{\mathrm{p}}(\tau_t^{\mathrm{p}}(\mathbf{r}_1), \ldots, \tau_t^{\mathrm{p}}(\mathbf{r}_m))$$

Snapshot reducibility states that a probabilistic snapshot of the result of an m-ary TP operation $op^{\mathrm{Tp}}(\mathbf{r}_1, \ldots, \mathbf{r}_m)$ at any time point t is equivalent to the result derived from the corresponding probabilistic operation op^{p} on the probabilistic snapshots of the input relations at t. Applying an atemporal operation over all probabilistic snapshots thus is consistent with snapshot reducibility in temporal databases and implies that the result at any time point t, both in terms of probability values and facts, is determined only by the input tuples that are valid at t. The application of op^{p} guarantees that the computations at each time point will yield Boolean lineage expressions that are consistent with the possible-worlds semantics [STW08, BSH$^+$08].

As example, consider the query of Fig. 2.1b over the relations of Fig. 2.1a. According to the lineage expression of tuple ('milk', [2,4), $c_1 \wedge \neg a_1$, 0.42), at $t = 2$, the fact 'milk' has been derived from the input tuples a_1 and c_1, i.e., the only input tuples of the probabilistic snapshot at $t = 2$ (Fig. 2.5 that include the fact 'milk'. Since the probability of 'milk' at $t = 2$ is only affected by the probabilities of a_1 and c_1, it can be computed based on the lineage expression $c_1 \wedge \neg a_1$.

Definition 2. (TP Change Preservation) *Let $\mathbf{r}_1, \ldots, \mathbf{r}_m$ be a set of TP relations, let op^{Tp} be an m-ary temporal-probabilistic operator, and let $u.T_s$, $u.T_e$ denote the start and end points of an interval associated with a tuple u. For each tuple $u \in \mathbf{u}$, where $\mathbf{u} = op^{Tp}(\mathbf{r}_1, \ldots, \mathbf{r}_m)$, it holds that:*

$$\forall t, t' \in u.T\,(\lambda_t^{\mathbf{u},u.F} \equiv \lambda_{t'}^{\mathbf{u},u.F}) \wedge$$
$$\nexists u' \in \mathbf{u}((u'.T_e = u.T_s \vee u'.T_s = u.T_e) \wedge (u'.\lambda \equiv u.\lambda))$$

Intuitively, change preservation ensures that only consecutive time points of tuples with equivalent lineage expressions are grouped into intervals. For example, the output tuples ('milk', [1,2), c_1, 0.6) and ('milk', [2,4), $c_1 \wedge \neg a_1$, 0.42) are not merged into the interval $[1,4)$, since they do not have equivalent lineages. Change preservation guarantees that a fact is valid over the same possible worlds with maximal intervals. The first line of Def. 2 ensures that the lineage expression at all time points in the interval of a result tuple is the same. The second line ensures that the time intervals produced by coalescing time points with the equivalent lineage expressions are maximal.[2]

2.5 TP Set Operations & Queries

2.5.1 TP Set Operations

In TP databases, the result of a *TP set union* includes, at each time point $t \in \Omega^T$, the facts for which there is a non-zero probability to be in **r** or in **s**; the result of a *TP set intersection* includes, at each time point, the facts for which there is a non-zero probability to be in **r** and in **s**; and the result of a *TP set difference* between two TP relations **r** and **s** includes, at each time point, the facts for which there is a non-zero probability to be in **r** and not in **s**.

[2]Rather than performing logical equivalence checks among Boolean formulas, which are co-NP-complete, we resort to a syntactic comparison of the lineage sets in our implementation.

Definition 3. (TP Set Operations) *Let* $\mathbf{r}$ *and* $\mathbf{s}$ *be temporal-probabilistic relations with schema* (F, λ, T, p)*, and let* $\lambda_t^{\mathbf{r},f}$ *denote the lineage expression of the tuple in relation* $\mathbf{r}$ *that includes fact* f *and is valid at time point* t*. Given a result tuple* $\tilde{r}$ *and the lineage-concatenation functions depicted in Table 3.3, we define the three TP set operations* $\mathbf{r} \cup^{Tp} \mathbf{s}$*,* $\mathbf{r} \cap^{Tp} \mathbf{s}$ *and* $\mathbf{r} -^{Tp} \mathbf{s}$ *as follows:*

$$\begin{aligned}
\tilde{r} \in \mathbf{r} \cup^{\text{Tp}} \mathbf{s} \iff \forall t \in \tilde{r}.T((\lambda_t^{\mathbf{r},\tilde{r}.F} \neq \texttt{null} \ \vee \ \lambda_t^{\mathbf{s},\tilde{r}.F} \neq \texttt{null}) \wedge \\
\tilde{r}.\lambda \equiv \mathbf{or}(\lambda_t^{\mathbf{r},\tilde{r}.F}, \lambda_t^{\mathbf{s},\tilde{r}.F})) \wedge \\
\forall t' \notin \tilde{r}.T(\tilde{r}.\lambda \not\equiv \mathbf{or}(\lambda_{t'}^{\mathbf{r},\tilde{r}.F}, \lambda_{t'}^{\mathbf{s},\tilde{r}.F}))
\end{aligned}$$

$$\begin{aligned}
\tilde{r} \in \mathbf{r} \cap^{\text{Tp}} \mathbf{s} \iff \forall t \in \tilde{r}.T(\lambda_t^{\mathbf{r},\tilde{r}.F} \neq \texttt{null} \wedge \lambda_t^{\mathbf{s},\tilde{r}.F} \neq \texttt{null} \wedge \\
\tilde{r}.\lambda \equiv \mathbf{and}(\lambda_t^{\mathbf{r},\tilde{r}.F}, \lambda_t^{\mathbf{s},\tilde{r}.F})) \wedge \\
\forall t' \notin \tilde{r}.T(\tilde{r}.\lambda \not\equiv \mathbf{and}(\lambda_{t'}^{\mathbf{r},\tilde{r}.F}, \lambda_{t'}^{\mathbf{s},\tilde{r}.F}))
\end{aligned}$$

$$\begin{aligned}
\tilde{r} \in \mathbf{r} -^{\text{Tp}} \mathbf{s} \iff \forall t \in \tilde{r}.T(\lambda_t^{\mathbf{r},\tilde{r}.F} \neq \texttt{null} \wedge \\
\tilde{r}.\lambda \equiv \mathbf{andNot}(\lambda_t^{\mathbf{r},\tilde{r}.F}, \lambda_t^{\mathbf{s},\tilde{r}.F})) \wedge \\
\forall t' \notin \tilde{r}.T(\tilde{r}.\lambda \not\equiv \mathbf{andNot}(\lambda_{t'}^{\mathbf{r},\tilde{r}.F}, \lambda_{t'}^{\mathbf{s},\tilde{r}.F}))
\end{aligned}$$

Table 2.1: Definition of lineage-concatenation functions.

$\textbf{\textit{and}}(\lambda_1, \lambda_2)$	$=$	$(\lambda_1) \wedge (\lambda_2)$	
$\textbf{\textit{andNot}}(\lambda_1, \lambda_2)$	$=$	$\begin{cases} (\lambda_1) \\ (\lambda_1) \wedge \neg(\lambda_2) \end{cases}$	$\begin{array}{l} \text{if } \lambda_2 = \texttt{null} \\ \text{otherwise} \end{array}$
$\textbf{\textit{or}}(\lambda_1, \lambda_2)$	$=$	$\begin{cases} (\lambda_1) \\ (\lambda_2) \\ (\lambda_1) \vee (\lambda_2) \end{cases}$	$\begin{array}{l} \text{if } \lambda_2 = \texttt{null} \\ \text{if } \lambda_1 = \texttt{null} \\ \text{otherwise} \end{array}$

The above definition of TP set operations specifies the intervals and lineage expressions of a result tuple $\tilde{r}$. The first line of the definition of each operation relates to Def. 1. It states that, at any time point $t \in \tilde{r}.T$, fact $\tilde{r}.F$ must be included in the corresponding input tuples from $\mathbf{r}$ and $\mathbf{s}$. Consequently, the lineage expression of the output tuple $\tilde{r}$ at each time point $t \in \tilde{r}.T$ (cf. second line) is computed based on the same input tuples, according to the lineage-concatenating functions of Table 3.3. In the case of set union, there must exist at least one tuple in either one of the two input relations that also includes $\tilde{r}.F$ over $\tilde{r}.T$. For set intersection, there must exist corresponding tuples in both input relations. For set difference, an output tuple is produced at

all time points t, at which there exists a tuple of the left relation r that is valid at t in $r.T$. This happens in two cases: (a) if a fact f is included in a tuple of **r** but in no tuple in **s**, and (b) if a fact f is included in a tuple of **r** but, with a probability of less than 1, also in a tuple of **s**. The first case resembles the definition of temporal set difference, where, at each time point in the output, there exist facts that are included in tuples of **r** and not in tuples of **s**. The second case occurs due to the probabilistic dimension. The result of a probabilistic set difference between **r** and **s** includes all facts, which have a non-zero probability to be in **r** and not in **s**.

Example 7. Figure 2.6 shows the relations **a** and **c** of Fig. 2.1a as well as selected output tuples of $\mathbf{a} -^{\text{Tp}} \mathbf{c}$. Different colors are used for different facts: green is used for 'milk', blue for 'dates' and red for 'chips'. Output tuples are drawn below the time axis. For example, the output tuple ('milk', $a_1 \wedge \neg c_2$, $[6,8)$, 0.09) satisfies Def. 3: for all time points in $[6,8)$, it holds that $\lambda_t^{\mathbf{a},\text{'milk'}} = a_1 \neq \texttt{null}$ and $\lambda_t^{\mathbf{c},\text{'milk'}} = c_2$. Thus, $\forall t \in [6,8)$, $\textbf{andNot}(\lambda_t^{\mathbf{a},\text{'milk'}}, \lambda_t^{\mathbf{c},\text{'milk'}}) \equiv a_1 \wedge \neg c_2$.

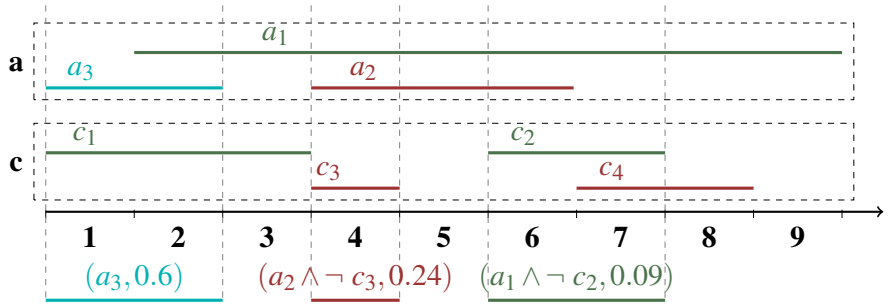


Figure 2.6: Selected output tuples of $\mathbf{a} -^{\text{Tp}} \mathbf{c}$.

The third line of the definition of each TP set operator is a direct consequence of Def. 2. It guarantees that, when merging consecutive time points into an interval, we consider only the ones for which the condition in the first line is satisfied. In other words, a new interval is created whenever there is a change in the validity of a tuple from either **r** or **s** at the currently considered time point. In Example 2.6, at time points $t = 5$ and $t = 8$, $\lambda_t^{\mathbf{a},\text{'milk'}} = a_1$ and $\lambda_t^{\mathbf{c},\text{'milk'}} = \texttt{null}$. Thus, outside the interval $[6,8)$ of tuple ('milk', [6,8), $a_1 \wedge \neg c_2$, 0.09), there are no time points for which $\textbf{andNot}(\lambda_t^{\mathbf{a},\text{'milk'}}, \lambda_t^{\mathbf{c},\text{'milk'}}) \equiv a_1 \wedge \neg c_2$. Fig. 2.7 shows the result of all TP set operations between relations **a** and **c** in Fig. 2.1a.

$\mathbf{a} \cup^{Tp} \mathbf{c}$

Product	λ	T	p
'milk'	c_1	[1,2)	0.6
'milk'	$a_1 \vee c_1$	[2,4)	0.72
'milk'	a_1	[4,6)	0.3
'milk'	$a_1 \vee c_2$	[6,8)	0.79
'milk'	a_1	[8,10)	0.3
'chips'	$a_2 \vee c_3$	[4,5)	0.94
'chips'	a_2	[5,7)	0.8
'chips'	c_4	[7,9)	0.8
'dates'	a_3	[1,3)	0.6

$\mathbf{a} -^{Tp} \mathbf{c}$

Product	λ	T	p
'milk'	$a_1 \wedge \neg c_1$	[2,4)	0.12
'milk'	a_1	[4,6)	0.3
'milk'	$a_1 \wedge \neg c_2$	[6,8)	0.09
'milk'	a_1	[8,10)	0.3
'chips'	$a_2 \wedge \neg c_3$	[4,5)	0.24
'chips'	a_2	[5,7)	0.8
'dates'	a_3	[1,3)	0.6

$\mathbf{a} \cap^{Tp} \mathbf{c}$

Product	λ	T	p
'milk'	$a_1 \wedge c_1$	[2,4)	0.18
'milk'	$a_1 \wedge c_2$	[6,8)	0.21
'chips'	$a_2 \wedge c_3$	[4,5)	0.56

Figure 2.7: TP set operations computed for the relations of Fig. 2.1a.

2.5.2 TP Set Queries & Complexity

Having defined TP set operations, we now move on to TP set queries, which are expressions of TP set operations over TP relations.

Definition 4. (TP Set Query) *Let* $\mathbf{r}_1, \ldots, \mathbf{r}_m$ *be duplicate-free TP relations. A* TP set query Q *is any expression of TP set operators that adheres to the following grammar:*

$$Q ::= \mathbf{r}_i \mid Q \cup^{Tp} Q \mid Q \cap^{Tp} Q \mid Q -^{Tp} Q \mid (Q)$$

The following theorem and corollary establish an interesting relationship between *safe queries* [DS07, DS12] in probabilistic databases and tractable queries in our TP setting. The theorem is based on the observation that repeated applications of TP set operations create regular lineage expressions, which are in *one-occurrence form* (1OF) [SOCK11] if none of the input relations occurs more than once in a TP set query. Formally, a formula is in 1OF iff no tuple identifier occurs more than once in the formula. Correspondingly, we call a TP set query Q *non-repeating* iff every input relation $\mathbf{r}_i$ occurs at most once in Q.

Theorem 1. Any non-repeating TP set query Q over duplicate-free TP relations yields lineage formulas in 1OF.

Proof. Consider a TP set operation over two TP relations $\mathbf{r}$ and $\mathbf{s}$, both having schema (F, λ, T, p). Since $\mathbf{r}$ and $\mathbf{s}$ are duplicate-free, we cannot have two tuples in either $\mathbf{r}$ or $\mathbf{s}$ that share the same fact at overlapping time intervals. Assume we have n_1 tuples in $\mathbf{r}$ and n_2 tuples in $\mathbf{s}$ with the same fact f, but each with non-overlapping time intervals. Then, for $n = n_1 + n_2$ input intervals, we can at most obtain $2n - 1$ output intervals. According to change preservation (Def. 2), we create the same amount of output tuples, one for each output interval and each with a different combination of tuple identifiers in their lineage (Def. 3). Next, inductively, during any further application of a TP set operation (over non-repeating subgoals), change preservation will only merge two consecutive time intervals iff their lineages are equivalent. This cannot occur, since all of the lineages that are created by an individual TP set operator are different. That is, for a non-repeating TP set query, each tuple identifier can occur at most once in the lineage of a result tuple, which means that the lineages are in 1OF. □

Corollary 1. *Any non-repeating TP set query Q over duplicate-free TP relations has PTIME data complexity.*

The proof of the corollary follows directly from Theorem 1, since computing the marginal probability of a Boolean formula in 1OF can be done in linear time in the size of the formula for independent random variables [SOCK11]. Also, all temporal alignment operations are of polynomial complexity (see [DBGJ16b, DBG12] as well as the algorithms in Section 2.7 and Section 2.8).

The above class of non-repeating TP set queries over duplicate-free TP relations nicely complements the dichotomy theorem [DS07, DS12] established for unions of conjunctive queries (UCQs) in probabilistic databases. Each individual TP set operation over two compatible relation schemas resolves to (a union of) at most two conjunctive queries, in which no intermediate duplicates due to a projection onto a subset of attributes in F may arise. Although repeated applications of TP set operations in a query do not necessarily form UCQs, the overall query remains hierarchical [SOCK11], since all attributes in F are propagated through the operations. Change preservation, on the other hand, which is required for a sequenced temporal semantics, preserves these complexity considerations by merging only intervals with equivalent lineage expressions into a single output interval. TP set queries with repeating subgoals however remain #P-hard as shown in [KRT11] (consider, e.g., the query $(\mathbf{r}_1 \cup^{\text{Tp}} \mathbf{r}_2) -^{\text{Tp}} (\mathbf{r}_1 \cap^{\text{Tp}} \mathbf{r}_3)$).

2.6 Lineage-Aware Temporal Windows

The result of all TP set operations includes facts whose probability is computed over maximal intervals, i.e., intervals during which the same input tuples are valid. The computation of such intervals in temporal databases is performed by adjusting the intervals of each input relation based on the tuples of the other input relation that are valid. Combining the adjusted results to identify the intervals when, for example, tuples of both relations are valid [DBG14], and concatenating their lineages for probability computation [DMT13, DBG14] must be performed with joins. In this section, we introduce the *lineage-aware temporal window*, a novel mechanism that directly associates candidate output intervals with the lineage expressions of the valid input tuples of both relations. We show that a window contains all the information to produce an output tuple of a TP set operation op^{Tp}, and that the set of all windows is a common core based on which all set operations can be computed using simple filtering and lineage-concatenation functions.

A lineage-aware temporal window has schema (F, T, $\lambda_{\mathtt{r}}$, $\lambda_{\mathtt{s}}$). F is a fact included in tuples over interval T. $\lambda_{\mathtt{r}}$ and $\lambda_{\mathtt{s}}$ are the lineage expressions of the input tuples of the left input relation $\mathbf{r}$ and the right input relation $\mathbf{s}$, respectively, which are valid over $[\mathtt{winTs}, \mathtt{winTe})$ and include F.

Definition 5. *(Lineage-Aware Windows) Let* $\mathbf{r}$ *and* $\mathbf{s}$ *be TP relations with schema (F, λ, T, p). The set of lineage-aware windows* $\mathbf{W}(\mathbf{r},\mathbf{s})$ *of* $\mathbf{r}$ *with respect to* $\mathbf{s}$ *with schema (F, T,* λ_r*,* λ_s*) is defined as follows:*

$$\begin{aligned}\tilde{w} \in \mathbf{W} \Longleftrightarrow \forall t \in \tilde{w}.T(\ (\lambda_t^{\mathbf{r},\tilde{w}.F} \neq \mathtt{null} \ \vee\ \lambda_t^{\mathbf{s},\tilde{w}.F} \neq \mathtt{null}) \wedge \\ (\tilde{w}.\lambda_r = \lambda_t^{\mathbf{r},\tilde{w}.F} \wedge\ \tilde{w}.\lambda_s = \lambda_t^{\mathbf{s},\tilde{w}.F})\ ) \wedge \\ \forall t' \notin \tilde{w}.T(\tilde{w}.\lambda_r \neq \lambda_t^{\mathbf{r},\tilde{w}.F} \vee\ \tilde{w}.\lambda_s \neq \lambda_t^{\mathbf{s},\tilde{w}.F})\end{aligned}$$

For a window $\tilde{w}$ to be created over $\tilde{w}.T$, at least a tuple of one of the input relations must be valid (Line 1). Each window $\tilde{w}$ in $\mathbf{W}(\mathbf{r},\mathbf{s})$ spans over the interval or a subinterval of a tuple r in $\mathbf{r}$ or a tuple s in $\mathbf{s}$ that include the fact $\tilde{w}.F$ and as stated in the second line of the definition these tuples will determine $\tilde{w}.\lambda_r$ and $\tilde{w}.\lambda_s$ respectively. Finally, according to line 3 of Definition 5, the interval of window $\tilde{w}$ is a maximal subinterval of an input tuple. In other words, at every time point outside the $\tilde{w}.T$, either an input tuple that was valid over $\tilde{w}.T$ stops being valid or an input tuple that was not valid over $\tilde{w}.T$ starts being valid.

Example 8. In Fig. 2.8, the TP relations **a** and **c** of Fig. 2.1 are illustrated along with the lineage-aware temporal windows of these two relations. Different colors are used for different facts: green for 'milk', red for 'chips', and blue for 'dates'. A rectangle represents a window, filled in the color of the tuples including the corresponding fact. The window w_1 = ('milk', [1,2), c_1, `null`) is colored green since it includes the fact $w_1.F$ = 'milk'. It indicates that, over interval [1,2), fact 'milk' is included in tuple c_1 of relation **c** ($w_1.\lambda_r = c_1$) but in no tuple of relation **a** ($w_1.\lambda_s$ = `null`). The window w_1 only spans the maximal interval [1,2), since at time point $t = 2$, tuple a_1 starts being valid and thus, there is a change in the tuples of the two relations that are valid at $t = 2$ and include fact 'milk'.

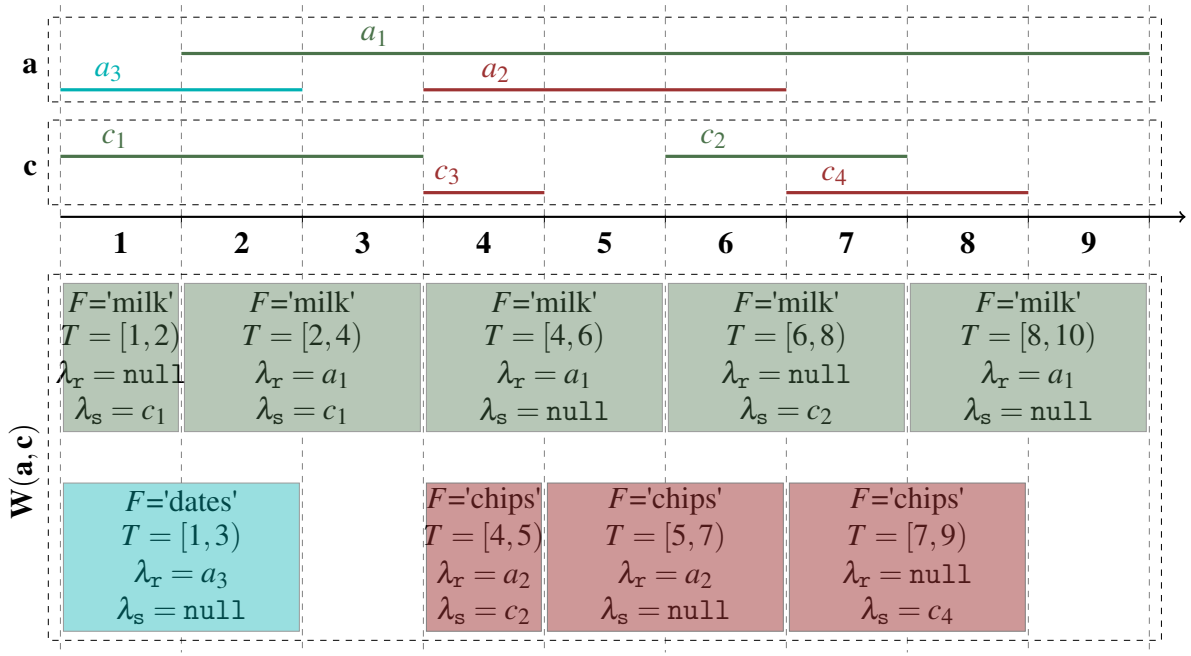


Figure 2.8: Lineage-Aware Temporal Windows $\mathbf{W}(\mathbf{a},\mathbf{c})$

Theorem 2. Let **r** and **s** be TP relations with schema (F, λ, T, p), op^{Tp} a TP set operation, and $\mathbf{W}(\mathbf{r},\mathbf{s})$ the lineage-aware windows of **r** and **s**. Given the output of the TP set-operation **r** op^{Tp} **s**, there exists a window w in W that contains all the necessary information to produce a tuple u in **r** op^{Tp} **s**.

Proof. We assume that op^{Tp} is a TP set-intersection ($\cap^{\text{Tp}}$) and u is an output tuple in **r** $\cap^{\text{Tp}}$ **s**. According to the definition of this operation and since, at each time point, only one tuple of each relation can include a fact, at each time point in $u.T$, there is exactly one tuple of **r** and one **s** valid and include $u.F$. Each window in $\mathbf{W}(\mathbf{r},\mathbf{s})$ records, for each fact F and time point t, the

tuples of each relation that include F at t. Thus, windows are only created over time points when there is at least one valid input tuple. In order for u to map to at least one window $w \in \mathbf{W}$, there must exist a window w with the same fact ($u.F = w.F$) and interval ($u.T = w.T$) as u, and for which it holds that $w.\lambda_r = \lambda_t^{\mathbf{r},u.F}$ and $w.\lambda_s = \lambda_t^{\mathbf{s},u.F}$. Assuming that there is no such window, i.e., assuming that one of the above mentioned conditions is not satisfied, we conclude that there are no valid tuples including $u.F$ or the interval $u.T$ is not maximal. This contradicts our initial assumption of u being a valid output tuple and of exactly one tuple of $\mathbf{r}$ and one $\mathbf{s}$ being valid over $u.T$ and including $u.F$. Consequently, there is at least one window $w \in \mathbf{W}$ to which we can map u. In turn, we assume that u maps to two windows w_1 and w_2 of $\mathbf{W}$. This means that u has the same fact and interval with both w_1 and w_2 and that $w_1.\lambda_r = \lambda_t^{\mathbf{r},u.F} = w_2.\lambda_r$ and $w_1.\lambda_s = \lambda_t^{\mathbf{s},u.F} = w_2.\lambda_s$. Consequently, window w_1 coincides with w_2, and this proves that there is exactly one window $w \in \mathbf{W}$ that contains all the information needed to produce an output tuple u for TP set-intersection. Similarly, we can prove that the same holds for an output tuple of any TP set operation. □

The flexibility of *lineage-aware temporal windows* relies on two characteristics: the lineages of valid tuples of each input relation are directly associated with a maximal interval, and they are separately recorded. These two characteristics allow for an efficient computation of the output tuples by using simple filtering conditions and lineage-concatenating functions instead of the additional joins performed in related approaches [DMT13, DBG14]. Given a TP set operation, $\lambda_{\mathtt{r}}$ and $\lambda_{\mathtt{s}}$ can be used to determine whether fact F and interval $[\mathtt{winTs}, \mathtt{winTe})$ yield an output tuple. If this is the case, $\lambda_{\mathtt{r}}$ and $\lambda_{\mathtt{s}}$ are combined to the lineage expression of this output tuple.

Theorem 3. Let $\mathbf{r}$ and $\mathbf{s}$ be TP relations with schema (F, λ, T, p), op^{Tp} a TP set operation, and $\mathbf{W}(\mathbf{r},\mathbf{s})$ the set of lineage-aware windows of $\mathbf{r}$ and $\mathbf{s}$. Given the filtering conditions λ_{filter} in Table 2.2 and the lineage-concatenating functions $\lambda_{function}$ of Definition 3, the computation of op^{Tp} is reduced to:

$$\mathbf{r}\ op^{\mathrm{Tp}}\ \mathbf{s}\ =\ \pi_{F,T,\lambda_{function}(\lambda_r,\lambda_s)}(\sigma_{\lambda_{filter}}(\mathbf{W}(\mathbf{r},\mathbf{s}))) \tag{2.2}$$

Proof. We assume that op^{Tp} is a TP set-intersection ($\cap^{\mathrm{Tp}}$), and a tuple u that is produced by the algebraic expression $\pi_{F,T,\mathit{and}(\lambda_r,\lambda_s)}(\sigma_{\lambda_r\neq\mathtt{null}\wedge\lambda_s\neq\mathtt{null}}(\mathbf{W}(\mathbf{r},\mathbf{s})))$. As a result, u has been produced from a window in $\mathbf{W}(\mathbf{r},\mathbf{s})$ for which $w.\lambda_r \neq \mathtt{null}$ and $w.\lambda_s \neq \mathtt{null}$. Also, $u.\lambda = \mathit{and}(w.\lambda_r, w.\lambda_s)$. Assuming that $u \notin \mathbf{r}\ \cap^{\mathrm{Tp}}\ \mathbf{s}$ means that one of the conditions in Def. 3 for TP set-intersection

Table 2.2: Definition of filtering conditions.

op^{Tp}	λ_{filter}	$\lambda_{function}$
$\mathbf{r} \cap^{Tp} \mathbf{s}$	$\lambda_r \neq \texttt{null} \wedge \lambda_s \neq \texttt{null}$	$\textbf{\textit{and}}(\lambda_r, \lambda_s)$
$\mathbf{r} -^{Tp} \mathbf{s}$	$\lambda_r \neq \texttt{null}$	$\textbf{\textit{andNot}}(\lambda_r, \lambda_s)$
$\mathbf{r} \cup^{Tp} \mathbf{s}$	$\lambda_r \neq \texttt{null} \vee \lambda_s \neq \texttt{null}$	$\textbf{\textit{or}}(\lambda_r, \lambda_s)$

is not satisfied. This is not possible, since u has been produced based on a window w and thus for all time points in $u.T$ or equivalently in $w.T$, $\lambda_t^{\mathbf{r},u.F} \neq \texttt{null}$, $\lambda_t^{\mathbf{s},u.F} \neq \texttt{null}$ and $u.\lambda = and(\lambda_t^{\mathbf{r},u.F}, \lambda_t^{\mathbf{s},u.F})$. Similarly, the contradiction can be shown for the time points outside $u.T$ and it can be shown that all tuples in $\mathbf{r} \cap^{Tp} \mathbf{s}$ are created based on the algebraic expression $\pi_{F,T,\lambda_{function}(\lambda_r,\lambda_s)}(\sigma_{\lambda_{filter}}(\mathbf{W}(\mathbf{r},\mathbf{s})))$. We can prove that the same holds for an output tuple of any TP set operation. □

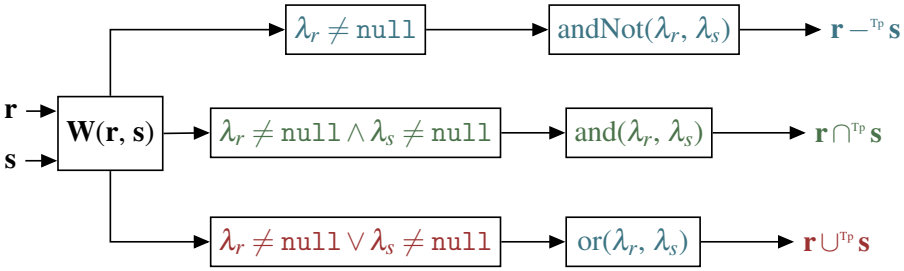


Figure 2.9: TP set operations using lineage-aware temporal windows.

In Theorem 3, we reduce the computation of a TP set operation $\mathbf{r}\ op^{Tp}\ \mathbf{s}$ to the application of a conventional projection and selection on the lineage-aware temporal windows of $\mathbf{r}$ and $\mathbf{s}$. The filtering condition in the selection as well as the lineage concatenating-function used in the projection are directly derived from the definition of TP set operations (Definition 3). The computation process is illustrated in Fig. 2.9. In comparison to existing temporal or probabilistic approaches used for set operations (cf. Fig. 2.2 and Fig. 2.3), the set of lineage-aware temporal windows constitutes a computational core that only needs to be computed once and does not suffer from the quadratic complexity of previous approaches, as shown in Section 2.7.

2.7 Lineage-Aware Window Advancer

In this section, we present the *lineage-aware window-advancer* (LAWA), an algorithm that produces all lineage-aware temporal windows of two TP relations. Each lineage-aware temporal window w in $\mathbf{W}(\mathbf{r},\mathbf{s})$ records the lineage expression of the tuple of each input relation that is valid over $w.T$ and that includes $w.F$. Since the interval of each window is maximal, a new window should be created when there is a change in the tuples of the input relations that are valid and include a given fact. Such a change only takes place when an input tuples starts or stops being valid, i.e., at the starting and ending points of input intervals, and this observation directly points to the use of a sweeping technique.

In our approach, to produce all lineage-aware temporal windows, we introduce LAWA, a sweeping algorithm we describe in Algorithm 1. Traditionally, sweeping algorithms use a vertical sweepline, and they determine the output tuples based on the input tuples that intersect with this sweepline [APR+98, PHD16]. This works well for TP set intersection. However, for TP set difference and set union, there are cases when the interval of an output tuple is not determined only by the tuples that intersect with the sweepline. In order to handle such cases, we use a *sweeping window*. The left and right boundaries of the window correspond to the start and end points of a maximal interval that is associated with a potential output interval.

LAWA processes the tuples of two duplicate-free TP relations $\mathbf{r}$ and $\mathbf{s}$ with schema (F, λ, T, p) that are sorted by their facts and starting points of their intervals. It produces lineage-aware temporal windows whose left (`winTs`) and right (`winTe`) boundaries are computed during a sweep of the start (`Ts`) and end (`Te`) points of the tuples. The left boundary $\texttt{winTs}_i$ of a window i is greater or equal to $\texttt{winTe}_{i-1}$ of the previous window. Its right boundary $\texttt{winTe}_i$ is the smallest among the end points of the tuples expected to overlap with this window, i.e., tuples with $\texttt{Ts} \leq \texttt{winTs}$ and $\texttt{Te} > \texttt{winTs}$, and the start points of the tuples of the two relations to be processed next.

The input of LAWA is a structure (`status`) with the necessary status information: the right boundary of the last candidate window (`prevWinTe`), the fact that is currently being processed (`currFact`), the current tuples of $\mathbf{r}$ (`rValid`) and $\mathbf{s}$ (`sValid`) that are valid over the sweeping window $[\texttt{winTs},\texttt{winTe})$, and the next tuples of relations $\mathbf{r}$ (`r`) and $\mathbf{s}$ (`s`). All variables are initialized to `null` except for `r` and `s` that are initialized to the first tuples of the corresponding relations. The value of `prevWinTe` is initialized to -1.

Algorithm 1: LAWA(status)

```
1  (prevWinTe, currFact, rValid, sValid, r, s) = status;
2  if rValid = null ∧ sValid = null then
3  |  if r = null ∧ s = null then                                   // Case 1
4  |  |  return (null, null)
5  |  else if r = null ∧ s ≠ null then                               // Case 2
6  |  |  winTs = s.Ts; currFact = s.F;
7  |  else if r ≠ null ∧ s = null then                               // Case 3
8  |  |  winTs = r.Ts; currFact = r.F;
9  |  else
10 |  |  if r.F = currFact ∧ s.F ≠ currFact then
11 |  |  |  winTs = r.Ts                                            // Case 4
12 |  |  if r.F ≠ currFact ∧ s.F = currFact then
13 |  |  |  winTs = s.Ts                                            // Case 5
14 |  |  else if r.Ts < s.Ts then                                   // Cases 6, 7
15 |  |  |  winTs = r.Ts; currFact = r.F;
16 |  |  else
17 |  |  |  winTs = s.Ts; currFact = s.F;
18 else winTs = prevWinTe ;                                         // Case 8
19 if r ≠ null ∧ r.F = currFact ∧ r.Ts = winTs then
20 |  rValid = r; r = getNext(r);
21 if s ≠ null ∧ s.F = currFact ∧ s.Ts = winTs then
22 |  sValid = s; s = getNext(s);
23 winTe = min(minTs(r, s), minTe(rValid, sValid));
24 λr = null; λs = null; window = null;
25 if rValid ≠ null then λr = rValid.λ;
26 if sValid ≠ null then λs = sValid.λ;
27 window = (currFact, winTs, winTe, λr , λs) ;
28 if rValid ≠ null ∧ rValid.Te=winTe then rValid = null;
29 if sValid ≠ null ∧ sValid.Te=winTe then sValid = null;
30 prevWinTe=winTe;
31 status = (rValid, sValid, r, s, currFact, prevWinTe);
32 return (window, status);
```

Initially, the left boundary `winTs` of the new window is determined, and the cases considered are described in Fig. 3.7. If at least one tuple is valid (Fig. 2.10h), the new window is adjacent to the previous one, with `winTs` = `prevWinTe` (Case 8, Line 18). Otherwise, `winTs`, and potentially `currFact`, are determined by the new tuples. Five possible scenarios exist: (a) both relations have been scanned (Case 1, Line 3), (b) one of the two relations has already been scanned (Cases 2 and 3, Lines 5– 8), (c) there are available tuples from both **r** and **s**, but only one includes the same fact as `currFact` (Cases 4 and 5, Lines10–13), (d) there are available tuples from both **r** and **s** and they either both include different facts from `currFact` or the same fact as `currFact`, making two starting points as candidates for *windTs* (Cases 6 and 7, Lines 14–17).

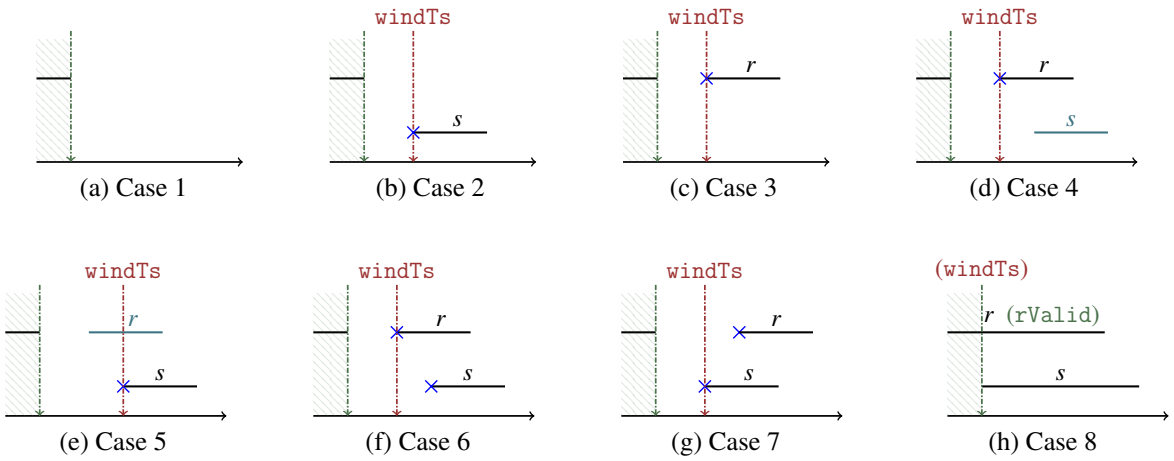


Figure 2.10: Cases for determining *windTs* in LAWA Algorithm. Blue crosses are used for the time points that are candidates for *windTs*.

Since the input relations are duplicate-free, i.e., no two tuples of the same relation can include the same fact and be valid at the same time point, `rValid` and `sValid` correspond to exactly one input tuple each. If `rValid` and `sValid` are not `null`, they correspond to tuples that were also overlapping with the previous window. Otherwise, they need to be updated to `r` or `s` if the latter include a fact equal to `currFact` and have a start point equal to `winTs` (Lines 19–22). The right boundary `winTe` is updated to the minimum time point among the end points of `rValid` and `sValid` and the current start points of `r` and `s`, i.e., the next tuples to be processed (Line 23). Here, the tuples `r` and `s` must be considered because the start point of an unprocessed tuple marks a change in the tuples that are valid over that interval.

After λ_r and λ_s are extracted from `rValid` and `sValid` (Lines 25–26), all the information for the creation of a lineage-aware temporal window is recorded (Line 22). `rValid` and `sValid`

are updated for the next call of LAWA based on whether the tuples they correspond to are still valid outside the window, i.e., when the end points of these tuples are larger than `winTe`. Finally, LAWA also returns its `status`, which is used in the implementation of the actual TP set operations.

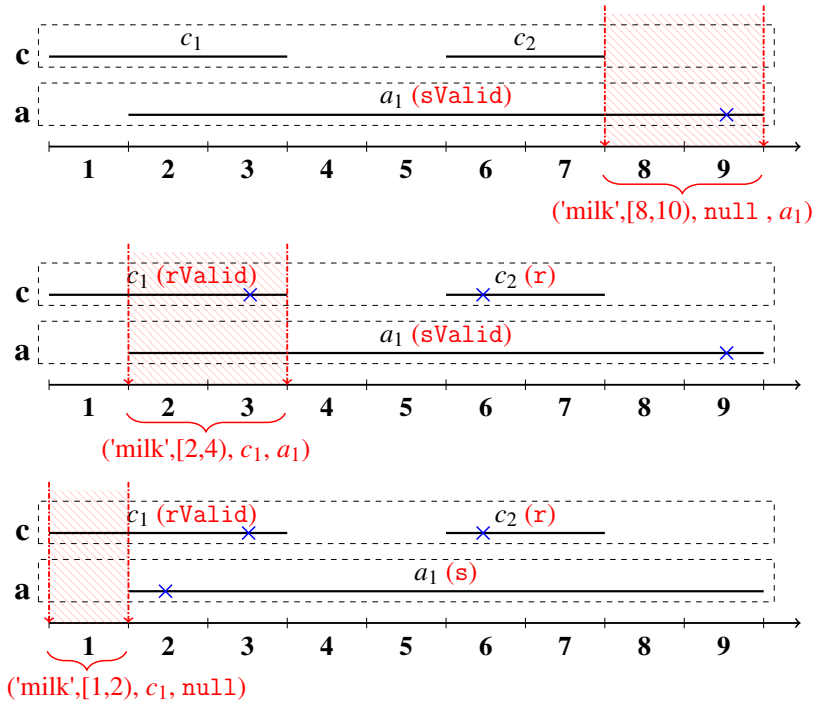


Figure 2.11: Three calls of LAWA for the input relations **c** and **a**.

Example 9. In Fig. 2.11, we illustrate three calls of LAWA with the left and right relations being **c** and **a** of Fig.2.1a, respectively. Before the first call, the input relations have been sorted by their facts and start points. The time points used to determine the right boundary of a window are annotated with a blue cross. In the first call of LAWA, illustrated at the bottom, the left and right boundary of the window are set to $\texttt{winTs} = 1$ and $\texttt{winTe} = 2$, respectively. After `winTs` is determined, the only tuple valid is $\texttt{rValid} = c_1$. Thus, given that there is no valid tuple in **a** yet, `winTe` is set to the start point of a_1, i.e., the next tuple of **a** to be processed. This time point is smaller than the end point $\texttt{Te} = 4$ of `rValid` or the start point $\texttt{Ts} = 6$ of the upcoming tuple of **c** (c_2). In the second call of LAWA, illustrated in the middle, the left boundary of the next window to be examined is equal to the right boundary of the previous window, i.e., $\texttt{winTs} = 2$, given that the fact ('milk') is still being processed. The tuples valid after time point $t = 2$ are $\texttt{rValid} = c_1$ and $\texttt{sValid} = a_1$. The right boundary of the window is the minimum of

rValid.Te = 4, sValid.Te = 10 and c_2.Ts = 6, and thus winTe = 4. A similar pattern goes on until the last call of LAWA, illustrated on the top of Fig. 2.11, where winTs = 8 and winTe = 10. Then, rValid and sValid are set to null and no further windows are produced.

2.8 Basic TP Set Algorithms

In this section, we implement all TP-set operations by exploiting the flexibility of *lineage-aware temporal windows* that enable finalizing output lineages and filtering out output intervals when they are produced, thus avoiding redundant computations that occur when these two steps are decoupled [DMT13, DBGJ16b]. Based on Theorem 3, we reduce the implementation of TP set operations into a four-step process (Fig. 2.12). The sorting step is a prerequisite for the creation of windows using LAWA. When a window is created, a lineage-based filter (λ_{filter}) is directly applied. The λ_{filter} is different for each TP set operation. In contrast to previous works of either temporal or probabilistic set operations, this step involves no application of additional algebraic operations, no tuple replication and no redundant interval comparisons. After the filtering step, the final lineage expression of an output tuple is created by applying the lineage-concatenating function ($\lambda_{function}$) of the respective TP set operation (Def. 3) on $\lambda_{\mathtt{r}}$ and $\lambda_{\mathtt{s}}$.

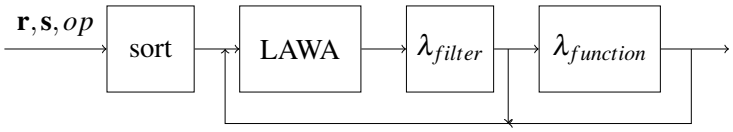


Figure 2.12: Process overview.

The algorithms *Intersect*(**r**, **s**), *Union*(**r**, **s**) and *Except*(**r**, **s**) correspond to $\mathbf{r} \cap^{\mathrm{Tp}} \mathbf{s}$, $\mathbf{r} \cup^{\mathrm{Tp}} \mathbf{s}$ and $\mathbf{r} -^{\mathrm{Tp}} \mathbf{s}$, respectively. In all algorithms, input relations are initially sorted based on their facts F and start points Ts (Line 2) when the status of LAWA is initialized. As long as the terminating condition (Line 8) is satisfied, LAWA passes through all start and end points in a smaller-to-larger fashion and produces candidate windows (Line 6). The windows produced by LAWA are filtered based on the lineages of the tuples that are valid during the interval it covers (Line 15). The filter used for each operation, as well as the terminating condition and the lineage-concatenating function, directly stem from the definitions of the operation. For example, in the case of set difference $\mathbf{r} -^{\mathrm{Tp}} \mathbf{s}$, windows are produced as long as there are tuples in the outer relation (i.e., while r ≠ null). The interval of a lineage-aware temporal window corresponds to an output

tuple only if there is a tuple of the outer relation that is valid over $[\texttt{winTs}, \texttt{winTe})$ (i.e., when $\lambda_\texttt{r} \neq \texttt{null}$).

For *Union*(**r**, **s**) and *Except*(**r**, **s**), when the while-loop terminates, there might still be one more window, corresponding to the subinterval of the last valid tuple of **r** (`rValid`) or the last valid tuple of **s** (`sValid`). Thus, LAWA is called one more time (Line 8).

Algorithm 2: *Intersect*(**r**, **s**)

```
1  sort(r{F,Ts}); sort(s{F,Ts});
2  status = (−1,null,null,null,fetchRow(r),fetchRow(s));
3  while status.r ≠ null ∧ status.s ≠ null do
4  |   (w,status) = LAWA(status);
5  |   if w.λr ≠ null ∧ w.λs ≠ null then
6  |   |   o = o ∪ {(F, and(w.λr, w.λs), [w.winTs, w.winTe))};
7  return o;
```

Algorithm 3: *Union*(**r**, **s**)

```
1  sort(r{F,Ts}); sort(s{F,Ts});
2  status = (−1,null,null,null,fetchRow(r),fetchRow(s));
3  while status.r ≠ null ∨ status.s ≠ null do
4  |   (w,status) = LAWA(status);
5  |   if w.λr ≠ null ∨ w.λs ≠ null then
6  |   |   o = o ∪ {(w.F, or(w.λr, w.λs), [w.winTs, w.winTe))};
7  if status.rValid ≠ null ∨ status.sValid ≠ null then
8  |   (w,status) = LAWA(status);
9  |   o = o ∪ {(w.F, or(w.λr, w.λs), [w.winTs, w.winTe))};
10 return o;
```

Algorithm 4: *Except*(**r**, **s**)

```
1  sort(r{F,Ts}); sort(s{F,Ts});
2  status = (−1,null,null,null,fetchRow(r),fetchRow(s));
3  while status.r ≠ null do
4  |   (w,status) = LAWA(status);
5  |   if w.λr ≠ null then
6  |   |   o = o ∪ {(w.F, andNot(w.λr,w.λs), [w.winTs, w.winTe))};
7  if status.rValid ≠ null then
8  |   (w,status) = LAWA(status);
9  |   o = o ∪ {(w.F, andNot(w.λr, w.λs), [w.winTs, w.winTe))};
10 return o;
```

Example 10. In Fig. 2.13, we illustrate the computation of set difference $\sigma_{\text{F = 'milk'}}(\mathbf{c}) -^{\text{TP}} \sigma_{\text{F = 'milk'}}(\mathbf{a})$ for relations **c** and **a** in Fig. 2.1a. The first candidate window $[1,2)$ has $\lambda_{\mathtt{s}} = \mathtt{null}$ and $\lambda_{\mathtt{r}} = c_1$. For set difference the current window yields a result tuple, since, over interval $[1,2)$, the fact ('milk') is included in a tuple of the left input relation **c** with lineage $\lambda_{\mathtt{s}} = c_1$. In contrast, the candidate ('milk', $[4,6)$, null, a_1) is rejected since ('milk') is not included in a tuple of the left input relation **c** over $[4,6)$.

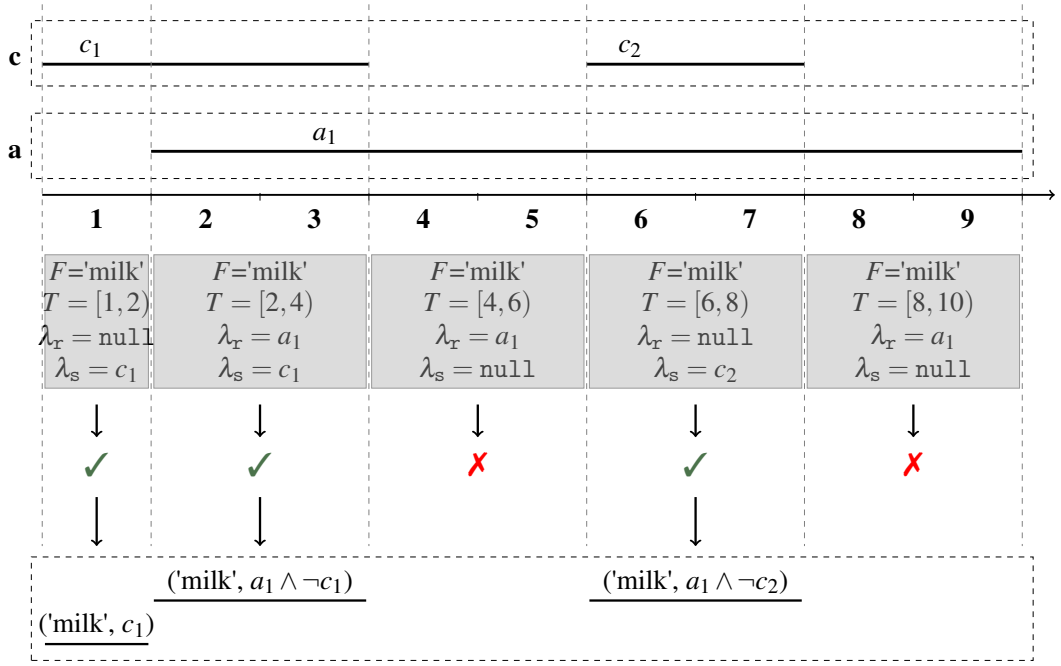


Figure 2.13: $\sigma_{\text{F = 'milk'}}(\mathbf{c}) -^{\text{TP}} \sigma_{\text{F = 'milk'}}(\mathbf{a})$

Time and Space Complexity: The time complexity of all TP set operations is determined by the complexity of the blocks presented in Fig. 2.12. Sorting has complexity $O(|\mathbf{r}|\log|\mathbf{r}| + |\mathbf{s}|\log|\mathbf{s}|)$ if it is comparison-based. A variant of counting-based sorting could also be used [KMV+13] (which is the case if Ω^T fits into main-memory), and in this case the corresponding complexity is even linear. After sorting, LAWA will sweep over all tuples in the sorted input relations **r** and **s**, accessing two input tuples at a time to determine the next window.

Proposition 1. Let **r**, **s** be two duplicate-free temporal-probabilistic relations. The upper bound of the number of windows produced by the window advancer is $n_r + n_s - f_d$ where n_r, n_s are the number of start and end points in **r** and **s**, and f_d is number of distinct facts in these relations.

By Proposition 1, the number of candidate windows considered by the algorithm is linear in the number of time intervals, and thus to the size of the input relations. Thus, LAWA has a time

complexity of $O(|\mathbf{r}| + |\mathbf{s}|)$, given that $|\mathbf{r}|$ and $|\mathbf{s}|$ are the numbers of tuples in the input relations $\mathbf{r}$ and $\mathbf{s}$, respectively. Moreover, the filtering and lineage-concatenation step for each candidate output tuple is performed in $O(1)$. Thus, the overall time complexity for computing TP set operations is $O(|\mathbf{r}|\log|\mathbf{r}| + |\mathbf{s}|\log|\mathbf{s}|)$, but may even be reduced to $O(|\mathbf{r}| + |\mathbf{s}|)$ if counting-based sorting is applicable. The use of *lineage-aware temporal windows* not only avoids the use for time-consuming additional operations for the filtering and lineage-concatenation steps, but also allows them to be performed directly at the time a window is created. That is, no intermediate buffers need to be maintained (apart from very few pointers), and thus the space complexity of all TP set operators is constant.

2.9 Experimental Evaluation

In this section, we evaluate LAWA in comparison to both temporal and temporal-probabilistic approaches that can be used for the computation of TP set operations. We perform experiments with real datasets as well as with synthetic datasets in which we vary (i) the number of facts in the input relations and (ii) the percentage of tuples whose intervals overlap. In all experiments, our approach empirically scales according to the bounds we provide in Section 2.8. LAWA is the only scalable approach that can be used for the computation of all three TP set operations, outperforming all state-of-the-art approaches for input relations of more than 10M tuples. In contrast to existing techniques, LAWA is robust, i.e., its performance behaves in a predictable manner with respect to the aforementioned characteristics of the datasets.

2.9.1 Experimental Setup

All of the following experiments were deployed on a 2xIntel(R) Xeon(R) CPU E5-24400 @2.40GHz machine with 64GB main memory, running CentOS 6.7. LAWA has been implemented in C++ [3], and all experiments were performed in main-memory. No indexes were used. In cases where PostgreSQL implementations were used, the maximum memory for sorting as well as for shared buffers was set to 1GB.

The TP set operations that different approaches can compute are presented in Table 2.3. Set difference is the least-supported operation, followed by set union and set intersection. Set in-

[3] http://www.ifi.uzh.ch/en/dbtg/research/tpset.html

Table 2.3: Approach Overview

Approach	$\mathbf{r} \cup^{Tp} \mathbf{s}$	$\mathbf{r} -^{Tp} \mathbf{s}$	$\mathbf{r} \cap^{Tp} \mathbf{s}$
LAWA	✓	✓	✓
NORM	✓	✓	✓
TPDB	✓	✗	✓
OIP	✗	✗	✓
TI	✗	✗	✓

tersection is the most-supported operation among the available systems, since it can be reduced to an interval join with an equality condition on the non-temporal attributes. Specifically, we compare our implementation of TP set operations using LAWA against:

Temporal-Probabilistic Database (TPDB) [DMT13]: The implementation of TPDB is an application connected with a DBMS and consists of three stages. The first stage parses Datalog rules with temporal predicates and translates them to SQL queries. The second stage executes the SQL queries in the DBMS. Base relations are stored in the DBMS, while lineage is kept as an internal data structure in main-memory. The third stage focuses on lineage processing by processing the base tuples with their Boolean connectives. We use the authors' original implementation, connected to PostgreSQL 9.4.3.

Normalize (NORM) [DBGJ16b]: The *Normalize* operator is implemented in the kernel of PostgreSQL by modifying its parser, executor and optimizer. We migrated the authors' implementation to PostgreSQL 9.4.3 for a fair comparison. To support TP set operations, we introduced reduction rules that are proper combinations of the temporal and probabilistic reduction rules (cf. [DBGJ16b, FO11]) and we illustrate them in Fig. 2.14.

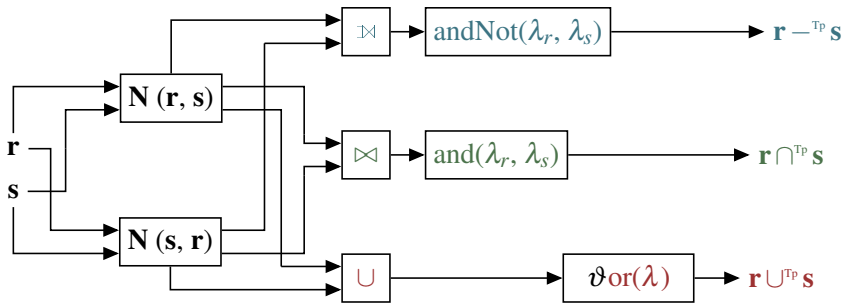


Figure 2.14: TP set operations using *NORM*. The approach adopted is a combination of the processes described in Fig. 2.2 and Fig. 2.3

Timeline Index (TI) [KMV+13]: This approach was used, in its original implementation, for the computation of TP set intersection, by applying a temporal join with an additional condition on the non-temporal attributes as well as the lineage-concatenating function ***and*** (see Table. 3.3).

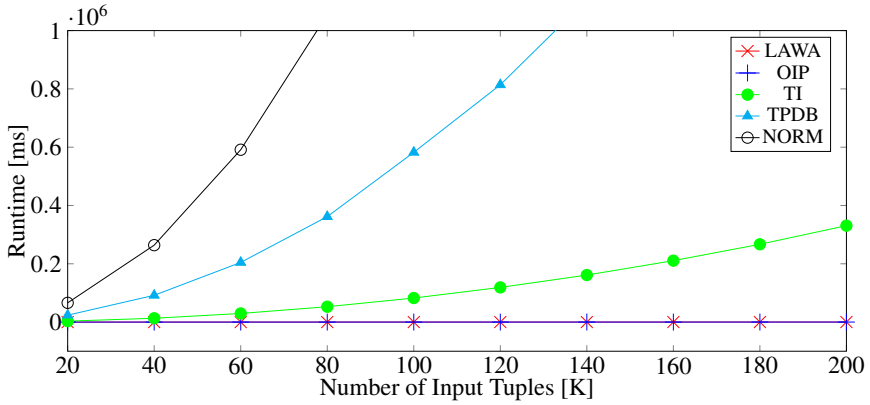
Overlap Interval Partition Join (OIP) [DBG14]: This approach is designed for overlap joins but does not support an additional filtering condition. For our experimental evaluation, we extended the authors' implementation, so that an equality condition on the non-temporal attributes of the tuples can be applied. In order to use OIP to compute set intersection, we first split each input relation into groups based on the facts included in each tuple. We then applied the OIP partitioning and join over each of these groups and merged the results.

2.9.2 Synthetic Dataset

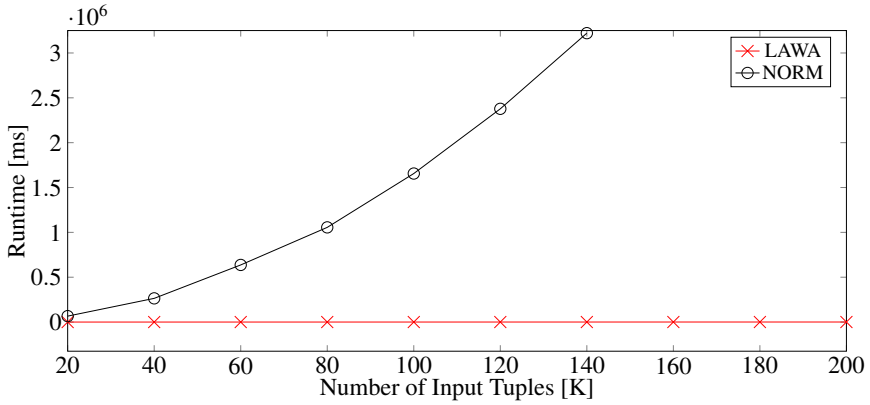
The parameters that we consider to populate a relation of our dataset are: (a) the length of the tuples' intervals, (b) the maximum time distance between two tuples that are consecutive and include the same fact, and (c) the number of different facts included in tuples of the relation. Assume all tuples of relations **r** and **s** have the same fact f. We define the *overlapping factor* of f as the number of maximal subintervals during which a tuple from **r** and **s** overlap, divided by the total number of maximal subintervals. Its value thus ranges in $[0,1]$. The higher the value of this metric, the more pairs of input tuples form output tuples, and therefore the more we stress-test the performance of the various approaches for TP set operations. According to Definition 3, overlapping time points are relevant for all set operations, whereas time points for which a fact is only included in the left input relation are only relevant for TP set difference.

1. Runtime. In the first setting, we fix the input tuples of all datasets to a single fact. We fix the overlapping factor to 0.6, and we randomly select the length of the intervals and the distance between two consecutive intervals in $[0,3]$. We then systematically increase the number of input tuples. In Fig. 2.15 and Fig. 2.16, we illustrate the performance of all the approaches for the computation of TP set operations for smaller datasets with up to 100K tuples and for larger datasets with up to 50M tuples, respectively.

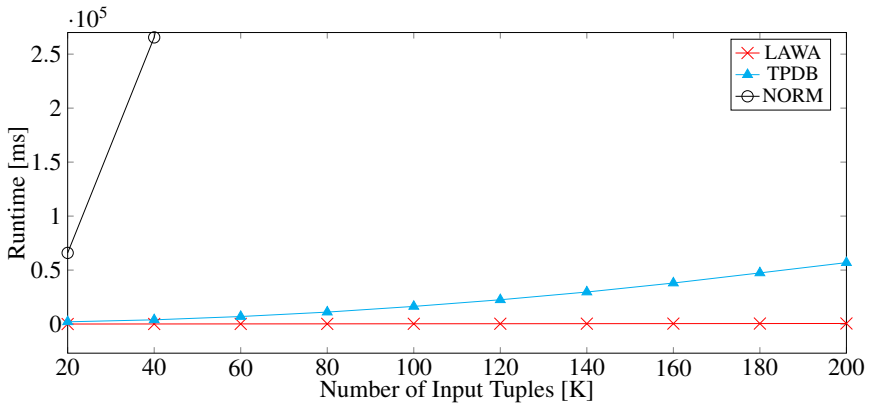
Smaller Datasets [20K–200K]: In Fig. 2.15, the datasets range from 20K to 200K tuples. Fig. 2.15a focuses on TP set intersection. The runtimes of LAWA and OIP hardly increase for the small datasets. Both outperform NORM, TI and TPDB by a large margin. OIP is specifically



(a) Set Intersection



(b) Set Difference



(c) Set Union

Figure 2.15: Synthetic Dataset [20K–200K]

designed for the computation of an overlap join, to which TP set intersection is reduced. NORM exhibits poor performance even if the number of input tuples is only 50K. In this approach, regardless of the operation, the two input relations need to first be normalized, such that, in their

adjusted versions, the intervals would be either equal or disjoint. The most expensive part of the normalization of a relation **r** using relation **s** is an outer join that uses inequality conditions on the start and end points to guarantee an overlap of the intervals. Although an additional inner join is applied in the case of TP set intersection, the performance of NORM suffers because of the outer join. Since all tuples include the same fact, but not all of them overlap, such a join has quadratic complexity [KLS+15].

In TPDB, queries are expressed using Datalog. Each rule may contain a conjunction of literals over the arithmetic predicates $=^T$, $\neq^T$ and $\leq^T$. In order to express TP set intersection, we use 6 reduction rules, one for each overlap relationship defined by Allen [All83]. TPDB then translates each rule to an inner join that is submitted to PostgreSQL. Although there is an equality condition on the non-temporal attributes, it is not used in the cases examined in Fig. 2.15 where all the tuples include the same fact. Thus, the joins are only based on the inequality conditions and perform a larger number of comparisons. TPDB is slower than the other approaches, but it is still faster than NORM, because the latter has to adjust each relation.

Although TI is faster than NORM and TPDB, it is one of the slowest approaches for set intersection. The index allows for the avoidance of redundant comparisons related to the interval overlap condition, and its creation cost is a small percentage of its runtime. Given the indexes of the input relations, TI performs a merge-join on them and produces (r_{id}, s_{id}) pairs. In order to form the output tuples, the input tuples corresponding to each pair need to be retrieved. Given the value of the overlapping factor and the existence of only one fact, a higher number of joined pairs is produced and thus a higher number of lookups is required. OIP splits the tuples of each input relation into partitions, based on the start/end points of their interval and its duration. Consequently, it offers a mechanism that performs interval comparisons between tuples only if their partitions overlap. If the partitions overlap, OIP performs a nested loop between the tuples of the two relations. As the overlapping factor is 0.6, which indicates that most of the pairs produced in the nested loop will indeed be output pairs, OIP has a very small percentage of false hits. Although OIP is tailored for an overlap join, for datasets of up to 200K tuples LAWA's performance is competitive, being on average 30 ms slower.

In the case of TP set difference, as illustrated in Fig. 2.15b, LAWA clearly outperforms NORM, for the same reasons as for TP set intersection. Fig. 2.15c compares LAWA with NORM and

TPDB during the computation of TP set union. LAWA has the lowest runtime, whereas NORM has the highest one, being 5 orders of magnitude slower than LAWA. The window that sweeps over all the input tuples in LAWA makes no false hits in this case, since all of the subintervals that the window defines correspond to output intervals. NORM no longer requires a join but a union after the relations have been normalized. However, as in all the previous operations, NORM's performance is hindered by the computation of the timestamp adjustment. TPDB can also compute TP set union by using a deduction rule that corresponds to a conventional union instead of joins, and thus its performance is significantly better in comparison to TP set intersection.

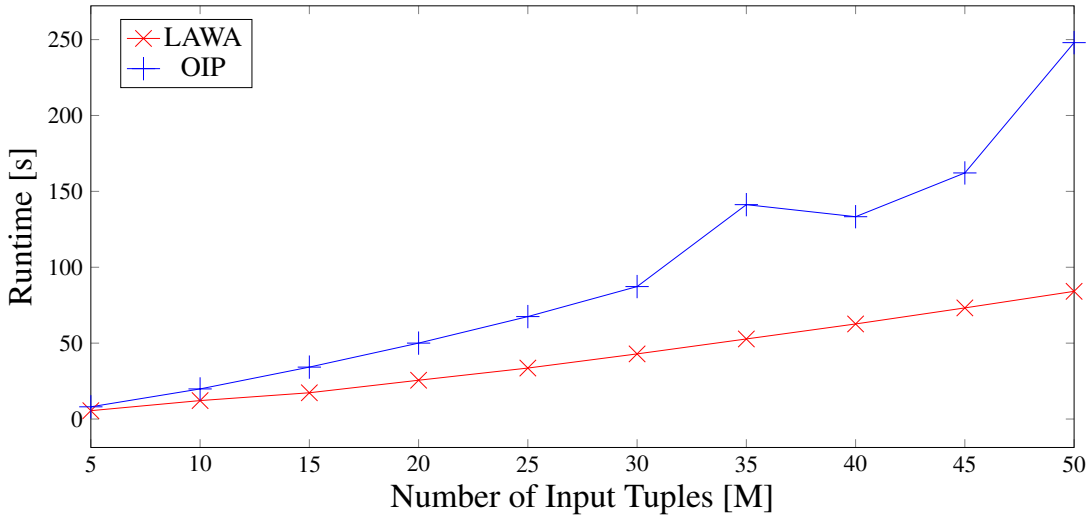


Figure 2.16: Synthetic Dataset [5M–50M]

Larger Datasets [5M–50M]: LAWA is the only scalable approach that can be used for the computation of all three TP set operations. In Fig. 2.16, we depict the performance of LAWA for the computation of TP set intersection for larger datasets. The overlapping factor of the datasets remains fixed to 0.6, and the dataset sizes vary from 5M to 50M tuples. While OIP is also considered, the other approaches that were included in Fig. 2.15a are not taken into consideration, since their runtimes were already two to five orders of magnitude higher when applied on the smaller datasets. After 30M tuples, LAWA is at least 2 times faster than OIP and continues to scale better. OIP produced a small number of partitions that contain many tuples each. Such partitions are likely to overlap and the nested loop that matches their tuples is computationally expensive. As far as TP set difference and TP set union are concerned, LAWA has similar runtime as in the case of TP set intersection and it is the only scalable approach suitable for their computation within at most 100 seconds.

2. Robustness. In this experiment, we show that LAWA is a scalable operator whose runtime only depends on the size of the dataset and not on its other characteristics (i.e., neither on the value of the overlapping factor nor on the number of distinct facts captured by the input tuples).

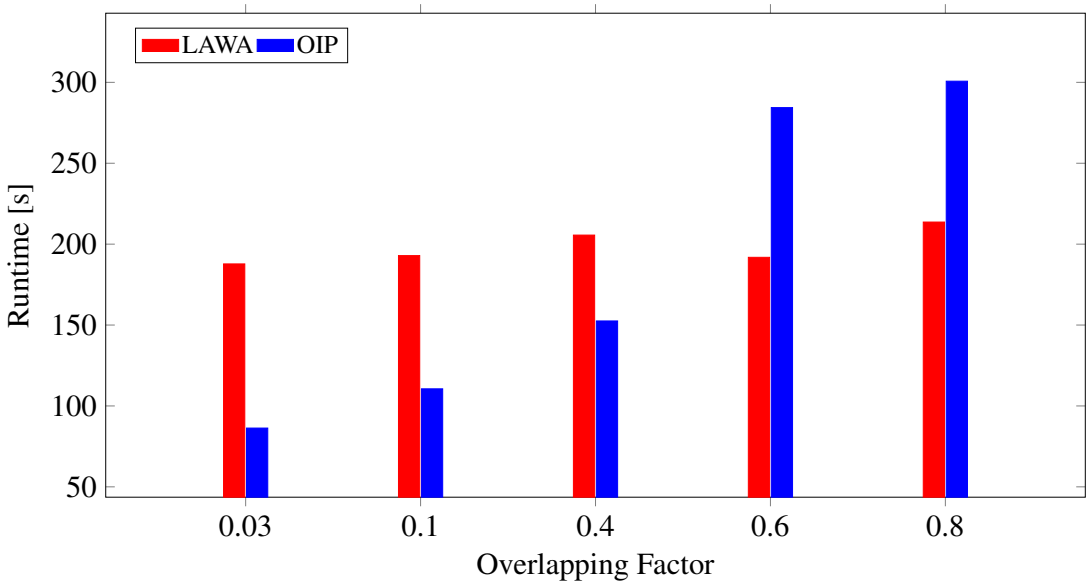
Table 2.4: Dataset Characteristics

Overlapping Factor	0.03	0.1	0.4	0.6	0.8
Max. Interval Length (R)	100	100	50	3	10
Max. Interval Length (S)	3	10	10	3	10
Max. Time Distance	3				

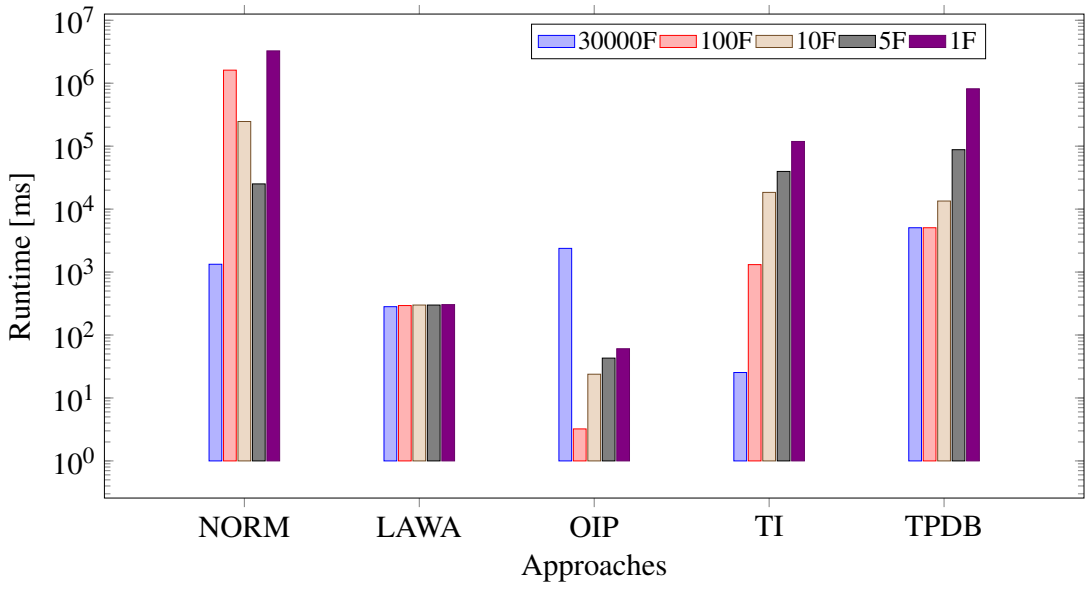
In Fig. 2.17a, the performance of LAWA for set intersection is compared with the one of OIP, which has been the most competitive approach for datasets where all the tuples include the same fact. This time, the size of the dataset is fixed to 30M, and the overlapping factor is assigned to four different values in $[0, 1]$. Table 2.4 depicts the *overlapping factor* of the datasets as well as their *maximum interval lengths* (in terms of the number of time points). The runtime of OIP increases as the overlapping metric increases. The reason is that the higher the overlapping factor, the more tuples occur in a partition and the nested loop performed in each partition is very time consuming. On the other hand, only minor variations are observed in the runtime of LAWA for the different values of the overlapping factor, thus demonstrating that the performance of LAWA is not negatively affected by interval-related characteristics of the dataset.

In Fig. 2.17b, we show how the number of distinct facts in the input relations affects the performance of LAWA and all other approaches during a TP set intersection. The size of the dataset is set to 60K, so that the runtimes of the approaches are comparable, and the overlapping metric is set to 0.6. The number of facts is set to values much less than the size of the dataset, but also to a value that is equal to half the size of the dataset. The runtime of LAWA remains stable as the number of the facts included in the input tuples decreases, whereas the performance of the other approaches deteriorates. OIP is an exception since, if the number of facts becomes comparable to the number of tuples, it suffers from the overhead of partitioning the tuples of each fact, performing the corresponding join and merging the results. Concerning the other approaches, TI has a better performance than LAWA but only in the case of 30K facts. This behaviour is expected, since there is a low number of joined pairs, thus reducing the number of required lookups. NORM's performance improves as well when the number of facts increases, but this approach

does not scale to datasets with more than 30K tuples. TPDB, on the other hand, appears to have diminishing improvements.



(a) Performance for varying overlapping factors.



(b) Performance for varying numbers of distinct facts.

Figure 2.17: Robustness Tests

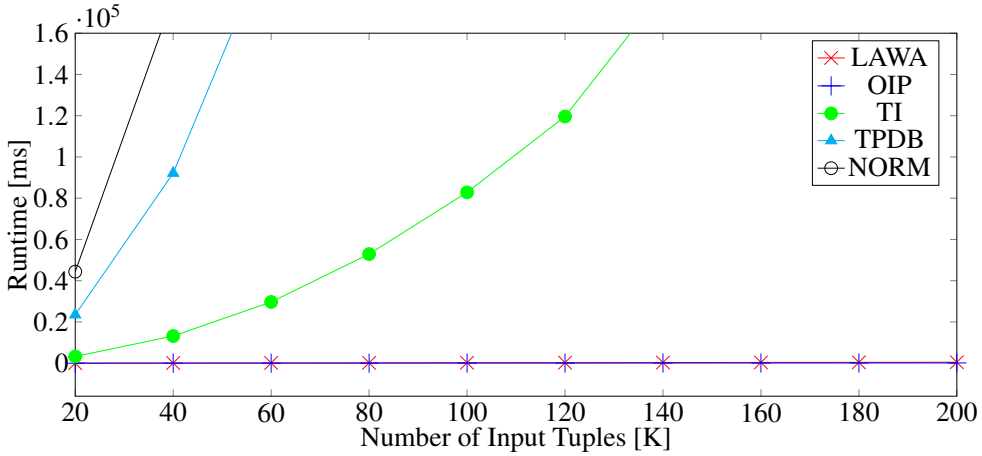
2.9.3 Real-World Datasets

In this subsection, we compare the runtimes of TP set operations using two real-world temporal datasets. The main properties of these datasets are summarized in Table 3.4. The Meteo Swiss dataset[4] includes temperature predictions that have been extracted from the website of the Swiss Federal Office of Meterology and Climatology. The measurements were taken at 80 different meteorological stations in Switzerland from 2005 to 2015. Measurements are 10 minutes apart and – in order to produce intervals – we merged time points whose measurements differ by less than 0.1. The Webkit dataset[5] [DBG14, PHD16, CB17] records the history of 484K files of the SVN repository of the Webkit project over a period of 11 years at a granularity of milliseconds. The valid times indicate the periods when a file remained unchanged. For both datasets we produced a second relation by shifting the intervals of the original dataset, without modifying the lengths of the intervals. The start/end points of the new relation were randomly chosen, following the distribution of the original ones.

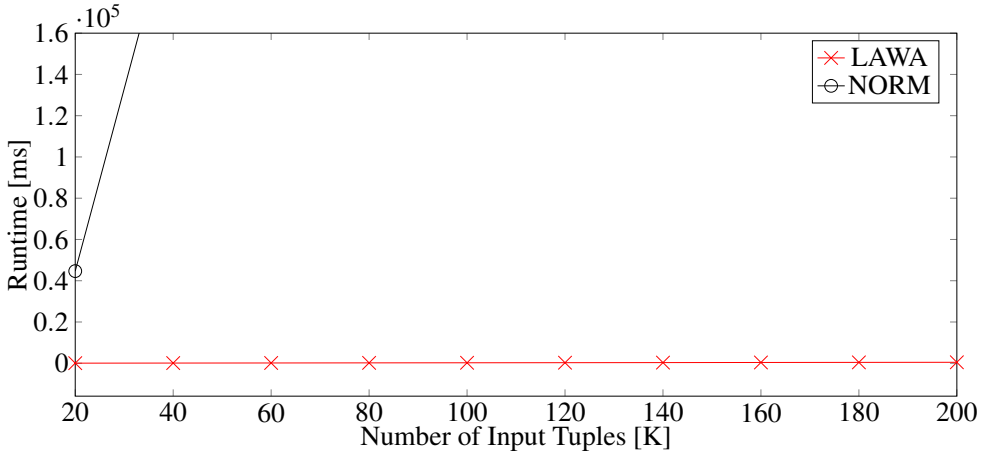
Table 2.5: Real-World Dataset Properties

	Meteo	**Webkit**
Cardinality	10.2M	1.5M
Time Range	347M	7M
Min. Duration	600	0.02
Max. Duration	19.3M	6M
Avg. Duration	152M	1.7M
Num. of Facts	80	484K
Distinct Points	545K	144K
Max Num. of Tuples (per time point)	140	369K
Avg Num. of Tuples (per time point)	37	21

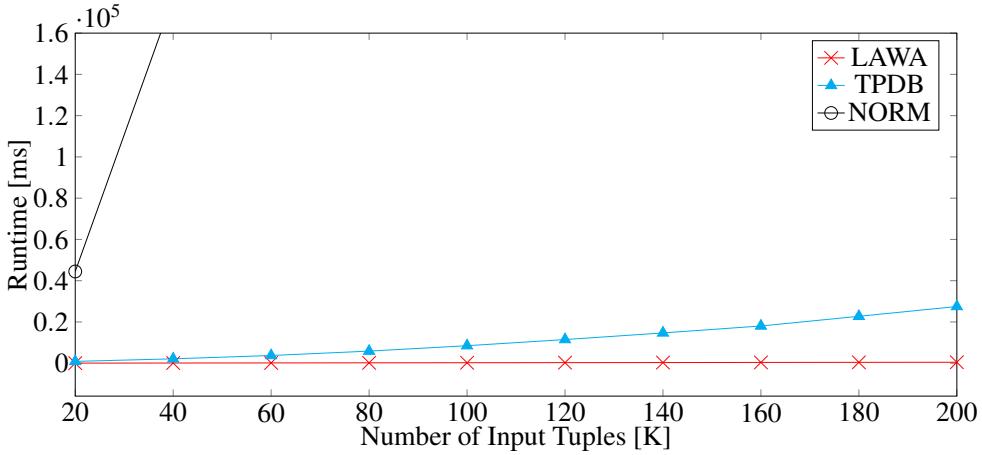
[4]Federal Office of Meteorology and Climatology: http://www.meteoswiss.ch (2016)
[5]The WebKit Open Source Project: http://www.webkit.org (2012)



(a) Set Intersection

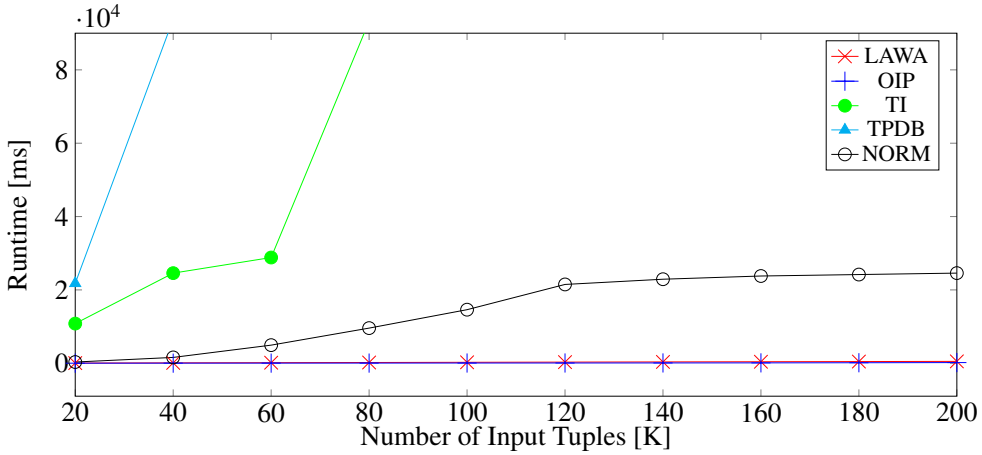


(b) Set Difference

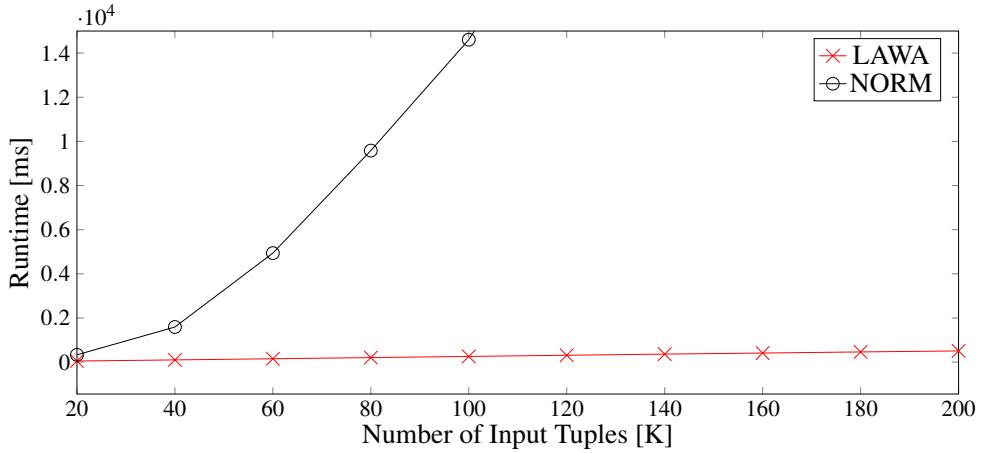


(c) Set Union

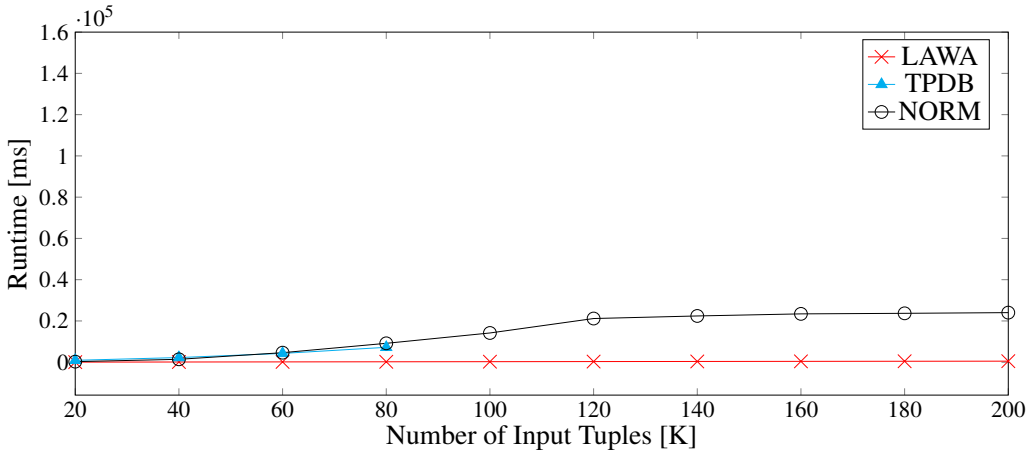
Figure 2.18: Meteo Swiss Dataset



(a) Set Intersection



(b) Set Difference



(c) Set Union

Figure 2.19: Webkit Dataset

In Fig. 2.18 and Fig. 2.19, we perform TP set intersection, difference and union over two equally sized relations created from random subsets of the initial dataset and its shifted counterpart, respectively. The runtime of each approach is based on the number of tuples in the input relations. In all cases, LAWA has the best performance. All approaches perform similarly to the synthetic dataset, with the exception of TI and NORM for the Webkit dataset. In this dataset, the maximum number of tuples starting or ending at a certain time point is very high, thus negatively affecting the performance of TI that has to make pairs among all of the tuples at a time point before it rejects the ones that do not match the nontemporal condition. Also, the number of facts is much higher than in the Meteo Swiss Dataset, making NORM significantly faster.

2.10 Conclusions

We proposed a novel data model that—for the first time in the literature—unifies the two areas of temporal and probabilistic databases under a sequenced semantics. We defined and implemented TP set operations, which can be supported very efficiently for a wide range of queries but received only very little attention so far. We introduced the lineage-aware temporal window as a mechanism to accelerate the computation of TP set operations. Our LAWA algorithm produces lineage-aware temporal windows that can be filtered directly by the time of their creation based on input lineage expressions. Using a generic window-sweeping technique, LAWA manages to produce all output intervals, not only for TP set intersection but also for TP set difference and TP set union, in a scalable and predictable manner. A thorough experimental evaluation reveals that our implementation is robust and outperforms comparable approaches from both temporal and probabilistic databases. As future work, we intend to investigate both tuple correlations and support for full relational algebra.

CHAPTER 3

Outer-Joins and Anti-Join in Temporal-Probabilistic Databases

Abstract

The result of a temporal-probabilistic operation with negation includes, at each time point, the probability with which a tuple of a positive relation **p** matches no tuple in a negative relation **n** under a predicate θ. Output tuples are produced when no matching tuple in **n** is valid or when all matching valid tuples in **n** are false. Moreover, TP outer-joins combine the characteristics of joins with and without negation and there exist time points, when the same input tuples produce two different result tuples since they have a non-zero probability to be $true$ or $false$. Existing solutions for TP joins either do not comply with the requirements for outer-joins and anti-joins or lead to redundant interval comparisons and to the recomputation of intermediate results. For the computation of TP joins with negation, we introduce generalized lineage-aware temporal windows, a mechanism that binds an interval to the lineages of all the matching valid tuples of each input relation. We group the windows of two TP relations into three disjoint sets based on the way that the attributes, lineage expressions and intervals are produced. By exploiting the

flexibility of lineage-aware temporal window, where the valid lineages of each relation remain decoupled, we compute all windows in an incremental manner. Pipelined computations not only alleviate the redundancies of existing approaches but also allow for the integration of our approach in PostgreSQL, as proven by an extensive experimental evaluation with real-world datasets.

3.1 Introduction

Operations with negation are performed for a positive relation **p**, a negative relation relation **n** and a θ condition that determines the tuples that match. In conventional databases, operations with negation disqualify an input tuple of the positive relation if its attributes match the attributes in a tuple of the negative relation. In temporal databases, the existence of a matching tuple in the negative relation does not disqualify the tuple of **p** itself but timepoints at which it is valid [BBJ98, BJ09]. In probabilistic databases, where tuples have a probability to be true or false, the existence of a matching tuple in **n** only reduces the probability with which a tuple is included in the output [Suc09, WRS08].

The result of a temporal-probabilistic operation with negation includes, at each time point, the probability with which a tuple of the positive relation **p** matches no tuple in the negative relation **n** under a predicate θ. Firstly, it includes output tuples that span subintervals when only tuples of the positive relation **p** are valid. In such cases, output intervals might be determined by starting or ending points of input tuples that are not valid during the output interval. Secondly, TP operations with negation produce outputs that indicate, at each time point, the probability of a tuple p_1 in the positive relation not matching any valid tuple in the negative relation because all of them are false. In this case, an output interval T is determined based on the starting and ending points of p_1 and of all the tuples of **n** that are valid over T and match p_1 under θ.

Example 11. Consider a booking website (Figure 3.1) that archives prediction data over time. Table **a** records data related to the locations that the clients want to visit, according to their searches. Table **b** records data regarding the availability of the hotels registered in the website, considering the busy periods in each location and on the rate at which each hotel gets booked. This archive corresponds to a temporal probabilistic database. Tuple ('Jim, WEN', a_2, [7,10), 0.8) captures that, at each day from the 7^{th} to the 10^{th} of the month, 'Jim wants to visit Wengen' with probability *0.8*. The website makes a prediction for each time point and there is no other tuple than a_2 in **a** that predicts the probability of 'Jim visiting Wengen' over an interval overlapping

a (wantsToVisit)

Name	*Loc*	λ	T	p
Ann	ZAK	a_1	[2,8)	0.7
Jim	WEN	a_2	[7,10)	0.8

b (hotelAvailability)

Hotel	*Loc*	λ	T	p
hotel_3	SOR	b_1	[1,4)	0.9
hotel_2	ZAK	b_2	[5,8)	0.6
hotel_1	ZAK	b_3	[4,6)	0.7

(a) Temporal-probabilistic base relations

$Q = \mathbf{a} \,\tilde{\bowtie}^{\text{Tp}}_{\theta}\, \mathbf{b},\ \theta : \mathbf{a}.\text{Loc} = \mathbf{b}.\text{Loc}$

Name	*Loc*	*Hotel*	λ	T	p
Ann	ZAK	-	a_1	[2,4)	0.70
Ann	ZAK	hotel_1	$a_1 \wedge b_3$	[4,6)	0.49
Ann	ZAK	hotel_2	$a_1 \wedge b_2$	[5,8)	0.42
Ann	ZAK	-	$a_1 \wedge \neg b_3$	[4,5)	0.21
Ann	ZAK	-	$a_1 \wedge \neg(b_3 \vee b_2)$	[5,6)	0.084
Ann	ZAK	-	$a_1 \wedge \neg b_2$	[6,8)	0.28
Jim	WEN	-	a_2	[7,10)	0.80

(b) Temporal-probabilistic tuple-based query

Figure 3.1: Temporal-probabilistic database example

with [7,10). In order to manage supply and demand, we determine the probability with which the client will find available accomodation at their preferred location, at each time point. The corresponding query is $Q = \mathbf{a} \,\tilde{\bowtie}^{\text{Tp}}_{\theta}\, \mathbf{b}$ ($\theta : \mathbf{a}.\text{Loc} = \mathbf{b}.\text{Loc}$), i.e., a temporal-probabilistic outer join with equality on the locations.

The answer tuple ('Ann, ZAK, hotel $_1$', $a_1 \wedge b_3$, [4,6), 0.49) expresses that, with probability 0.49, Ann will visit Zakynthos (a_1) and stay at hotel_1 in Zakynthos (b_3) during interval [4,6). It is valid over the intersection of the intervals of tuples a_1 and b_3 and it is true when both these tuples are true. Answer tuple ('Ann, ZAK, -', a_1, [2,4), 0.7) expresses that, with probability 0.7, Ann will visit Zakynthos (a_1) but there is no hotel available to stay there. Although the lineage and the output probability are both determined by tuple a_1, i.e., the only tuple valid during [2,4), the interval of this output tuple is influenced by the starting point of tuple b_3, a tuple not valid over [2,4) and not included in the output lineage. Over the interval, [5,6) there is 0.084 probability that Ann visits Zakynthos but finds no accommodation. According to answer tuple ('Ann, ZAK, - ', $a_1 \wedge \neg(b_3 \vee b_2)$, [5,6), 0.084), during [5,6), the output is influenced but more than a pair of input tuples. Although all tuples are valid over [5,6), this tuple is true when 'Ann visits Zurich'

(a_1 is true) but also when neither hotel_1 nor hotel_2 are available during [5,6) (b_3 and b_2 are false). Finally, there are time points when tuples b_2 and b_3 contribute to two output tuples depending on whether they are $true$ or $false$.

TP set-difference is the only temporal-probabilistic operation with negation that is currently supported [PTB18]. Since set-operations combine only tuples with pairwise equal non-temporal attributes, simplified structures can be used. Specifically, only one tuple of each relation is valid at each time point which allows for solutions with linearithmic complexity. For TP outer-joins and TP anti-join, the θ-condition is not restricted to equality and multiple tuples of the negative relation might be valid over an output interval and input tuples with non-temporal attributes that are not pairwise equal might be combined to form an output tuple. Moreover, TP outer-joins combine the characteristics of TP operations with and without negation: at each time point, two outcomes are possible since the same tuples can be either $true$ or $false$.

Temporal-probabilistic approaches that only compare pairs to compute joins cannot produce output tuples related with negation [DMT13]. Temporal approaches [DBGJ16b] require adjusting the input relations and recombining adjusted results to produce output facts and lineages. Adjusting input tuples and combining the adjusted results leads to redundant comparisons and to recomputation of intermediate results. Lineage-aware temporal windows [PTB18] have been introduced as a mechanism that binds an output interval with the lineages of the tuples that are valid during the interval. They overcome the redundancies in the decoupled computation of output intervals and lineages but are tailored to cases when one tuple of each input relation is valid and when the input tuples have the same non-temporal attributes. TP joins with negation must handle cases when multiple tuples of each relation might be valid over an output interval and when an output tuple is formed based on two tuples with different non-temporal attributes.

In this paper, we introduce *generalized lineage-aware temporal windows* and propose splitting all windows that can be produced by two TP relations into three disjoint sets: the unmatched windows, the overlapping windows and the negating windows. Each group covers a different case of output tuples and the result of every TP join that includes negation is expressed based on these three sets. We exploit the decoupled lineages of the valid tuples of each relation in a window to compute all windows in an incremental manner.

Outline & Contributions.

- We introduce *generalized lineage-aware temporal windows* so that we can produce output tuples for pairs of input tuples with different non-temporal attributes and for cases when

	F_r	F_s	λ_r	λ_s	T
w_1	'Ann, ZAK'	-	a_1	-	[2,4)
w_2	'Jim, WEN'	-	a_2	-	[7,10)

(a) Unmatched Windows

	F_r	F_s	λ_r	λ_s	T
w_3	'Ann, ZAK'	'hotel$_1$, ZAK'	a_1	b_3	[4,6)
w_4	'Ann, ZAK'	'hotel$_2$, ZAK'	a_1	b_2	[5,8)

(b) Overlapping Windows

	F_r	F_s	λ_r	λ_s	T
w_5	'Ann, ZAK'	-	a_1	b_3	[4,5)
w_6	'Ann, ZAK'	-	a_1	$b_3 \vee b_2$	[5,6)
w_7	'Ann, ZAK'	-	a_1	b_2	[6,8)

(c) Negating Windows

Figure 3.2: The three disjoint sets of lineage-aware temporal windows created based on the relations **a** and **b** in Fig. 3.1 and on the θ-condition a.Loc=b.Loc

multiple input tuples are valid. Given a θ-condition and two TP relations, we group windows into three disjoint sets: the *unmatched windows*, the *overlapping windows* and the *negating windows*. We show that the result of each TP binary operator can be expressed using these three sets and the appropriate lineage-concatenation functions.

- We introduce the algorithms LAWA_{U} and LAWA_{N} for the computation of unmatched and negating windows, respectively. Recording the lineages of the tuples valid in the left and right relation over an output interval and keeping them decoupled until the formation of output tuples, allows for the computation of unmatched and negating windows based on the overlapping ones. Thus, redundant interval comparisons due to the repetition of basic steps are avoided and the runtime required for the computation of outer-joins and anti-join improves by two orders of magnitude.
- We conduct extensive experiments using real datasets to compare our approach for the computation of TP outer-joins and TP anti-join with existing state of the art approaches. Our approach is integrated in PostgreSQL and exhibits a lower runtime while being scalable.

The remainder of this paper is organized as follows. Section 3.2 provides an overview of related works on temporal and probabilistic databases with a focus on outer-joins and anti-join. Section 3.3 discusses our TP data model and its query semantics. Section 3.4 discusses the impact

of negation in the result of temporal probabilistic operations. Section 3.5 generalizes lineage-aware temporal windows and introduces the three disjoint sets of windows that can be produced based on two TP relations. Section 3.6 expresses the result of TP joins that include negation using the sets of lineage-aware windows. Section 3.7 introduces two algorithms for the computation of the different window sets while section 3.8 presents a comprehensive performance study that compares our implementation with existing timestamp-adjusting and lineage-computation approaches. Section 3.9 finally concludes the paper.

3.2 Related Work

We review related approaches from temporal and probabilistic databases and explain their limitations in terms of supporting TP outer joins and anti-join.

Temporal-Probabilistic Operations. Dylla et al. [DMT13] introduced a closed and complete TP database model, coined TPDB, based on existing temporal and probabilistic models. Query processing is performed in two steps. The first step, grounding, evaluates a chosen deduction rule (formulated in Datalog with additional time variables and temporal predicates) and computes the lineage expressions of the deduced tuples. The second step, deduplication, removes the duplicates that could occur in the grounding step by adjusting intervals. The grounding step performs pairwise comparisons and thus subintervals that are present in one of the two input relations, i.e., during which no tuple of the other relation is valid, cannot be produced.

TP Operations with negation. Set-difference is the only TP operation with negation currently supported [PTB18]. For its computation, Papaioannou et al. introduced *l*ineage-aware temporal windows, a mechanism that binds an output interval with the lineage of the tuple in each input relations that includes a fact F and that is valid during the interval. The *l*ineage-aware temporal windows eliminate the redundant interval comparisons and additional joins for the formation of lineage expressions in TP set operations. The starting and ending points of the interval that the window spans are computed via a comparison of the starting and ending points of input tuples that are valid but also of neighboring tuples. Thus, they are appropriate for the formation of output intervals that do not correspond to the overlap of a pair of valid tuples. In TP joins with negation, input tuples including different non-temporal attributes are also combined and multiple tuples of an input relation can be valid over an interval and need to be included in the lineage of an output tuple.

Temporal Joins. In temporal databases, the result of a temporal outer-join op^T is defined as the result of applying op over a sequence of atemporal instances (the so-called snapshots) of the input relations—a key concept in temporal databases termed *snapshot reducibility* [AGC+13, LM97, VL07]. Maximal intervals are produced by merging consecutive time points to which the same input tuples have contributed (*change preservation*). Dignös et al. [DBG12, DBGJ16b] use *data lineage* to guarantee change preservation for all relational operations under a sequenced semantics. For the computation of joins, they introduce the *alignment* operator. Intuitively, the alignment $\Phi(\mathbf{r},\mathbf{s})$ of a relation $\mathbf{r}$ based on another relation $\mathbf{s}$ replicates the tuples of $\mathbf{r}$ and assigns new time intervals to them. The new intervals are obtained by splitting the original intervals of $\mathbf{r}$ based on tuples of $\mathbf{s}$ with which they overlap. The valid tuples of both relations that contribute to the adjustment of an interval are not maintained along with the new intervals. This is the reason why the alignment of both relations is required as well as the application of op to produce all output tuples [DBG12, DBGJ16b]. Using this approach in a TP context, other than the overhead and redundancy of aligning both relations, the input tuples must also be adjusted in groups and not only in pairs for the cases when valid tuples are $false$. Combining adjustment both in pairs and in groups multiple times in the same query incurs redundant comparisons and recomputation of intermediate results.

Sweeping-based approaches have been widely used for the computation of overlap joins [PHD16, APR+98] in temporal settings. A sweepline moves over all start and end points of tuples, and determines, for each time point, the tuples of both input relations that are valid. These approaches are tailored to compute efficiently the overlap join but are not suitable for the computation of the class of operations discussed in this paper. First, the overlapping intervals computed in these approaches only correspond to a part of the result of a TP outer-join while they are not included in the result of a TP anti-join. Second, they generally do not consider join conditions on the non-temporal attributes limiting the types of queries they could be used for.

Probabilistic Joins. In probabilistic databases, the result of a probabilistic operation op^p is defined as the result of applying op over the set of all possible instances of the input relations. The Trio system [STW08] was among the first to recognize *data lineage*, in the form of a Boolean formula, as a means to capture the possible instances at which an output tuple is valid. In an effort to provide a *closed and complete* representation model for uncertain relational data, they introduced *Uncertainty and Lineage Databases* (ULDBs) [BSH+08]. The algebraic operators are modified to compute the lineage of the result tuples in a ULDB, thus capturing all information needed for computing query answers and their probabilities. Fink et al. [FOR11, FO16]

reduced the computation of probabilistic algebraic operations to conventional operations so that these can be performed using a DBMS, rather than by an application layer built on top of it. In all these works, the focus is restricted to select-project-join queries. Probabilistic anti-joins expressed using the NOT EXISTS predicate in SQL have been explored by Wang et al. [WRS08]. They have been integrated in MystiQ by breaking the initial query into positive and negative subqueries that are separately evaluated and then combined. Incorporating interval computation with predicates in these approaches is possible but does not comply with all the requirements of TP operations with negation, similarly to the work of Dylla et al. [DMT13].

3.3 Background

We denote a **temporal-probabilistic schema** by R^{Tp} (F, λ, T, p), where $F = (A_1, A_2, \ldots, A_m)$ is an ordered set of attributes, and each attribute A_i is assigned to a fixed domain Ω_i. λ is a Boolean formula corresponding to a lineage expression. T is a *temporal attribute* with domain $\Omega^T \times \Omega^T$, where Ω^T is a finite and ordered set of *time points*. p is a *probabilistic attribute* with domain $\Omega^p = (0,1] \subset \mathbb{R}$. A **temporal-probabilistic relation r** over R^{Tp} is a finite set of tuples. Each tuple $r \in \mathbf{r}$ is an ordered set of values in the appropriate domains. The value of attribute A_i of r is denoted by $r.A_i$. The conventional attributes $F = (A_1, A_2, \ldots, A_m)$ of tuple r form a *fact*, and we write $r.F$ to denote the fact f captured by tuple r. For example, the base tuple ('Ann, ZAK', a_1, $[2,8)$, 0.7) of relation **a** (see Fig. 3.1a) includes the fact $a_1.F$ = ('Ann, ZAK'), the lineage expression $a_1.\lambda = a_1$, the time interval $a_1.T = [2,8)$, and the probability value $a_1.p = 0.7$. The temporal-probabilistic annotations of the schema express that (i) $a_1 = \mathit{true}$ with probability $a_1.p$ for every time point in $a_1.T$, (ii) $a_1 = \mathit{false}$ with probability $1 - a_1.p$ for every time point in $a_1.T$, (iii) and a_1 is always *false* outside $a_1.T$. By following conventions from [DMT13, DBGJ16b, DBG12, OHK09], we assume duplicate-free input and output relations. Formally, a temporal-probabilistic relation **r** is **duplicate-free** iff $\forall r, r' \in \mathbf{r}(r \neq r' \Rightarrow r.F \neq r'.F \vee r.T \cap r'.T = \emptyset))$. In other words, the intervals of any two tuples of **r** with the same fact f do not overlap.

A **lineage expression** λ is a Boolean formula, consisting of tuple identifiers and the three Boolean connectives $\neg$ ("not"), $\wedge$ ("and") and $\vee$ ("or"). Tuple identifiers represent Boolean random variables among which we assume independence [DMT13, OHK09, DS07]. For a base tuple r, $r.\lambda$ is an atomic expression consisting of just r itself. For a result tuple $\tilde{r}$ derived from one or more TP operations, $\tilde{r}.\lambda$ is a Boolean expression as defined above. For a result tuple,

lineage is determined by the temporal-probabilistic operators (formally defined in Section 3.6) that were applied to derive that tuple from the base tuples. The probability of a result tuple is computed via a probabilistic valuation of the tuple's lineage expression, using either exact (see, e.g., [DS07, DS12, OH08]) or approximate (see, e.g., [FHO13, FO11, GS14, GS15, OHK10]) algorithms. For example, in the result relation of Fig. 3.1b, the lineage $a_1 \wedge \neg b_3$ yields a marginal probability of $0.7 \cdot (1-0.7) = 0.21$ by assuming independence among the base tuples a_1 and b_3 (see Fig. 3.1a).

Further, we write $\lambda_t^{\mathbf{r},f}$ to refer to the disjunction of the lineage expressions of the tuples in relation $\mathbf{r}$ with fact f that are valid at time point t. We write $\lambda_t^{\mathbf{r},\theta}$ to refer to the disjunction of the lineage expressions of the tuples in relation $\mathbf{r}$ that satisfy θ and are valid at time point t. When there are no tuples in $\mathbf{r}$ with fact f or satisfying θ at time point t, we write $\lambda_t^{\mathbf{r},f} = \texttt{null}$ or $\lambda_t^{\mathbf{r},\theta} = \texttt{null}$, respectively. We write $\theta_{\tilde{r}}$ to indicate that values of attributes in condition θ are instantiated to the corresponding values in tuple $\tilde{r}$. For example, given the θ condition used in the query of Figure 3.1b and $\tilde{r}$ = ('Ann, ZAK, hotel$_1$', $a_1 \wedge b_3$, [4, 6), 0.49), $\theta_{\tilde{r}}$ is $b.Loc$ = 'ZAK'.

The query semantics of the sequenced TP data model is based on an intriguing analogy between the sequenced semantics of the temporal dimension and the possible world semantics of the probabilistic dimension: rather than iterating over snapshots or possible worlds, they both use the notion of data lineage to define their operational semantics. Given a TP relation $\mathbf{r}$, a tuple $r \in \mathbf{r}$ is valid at every time point t included in its time interval $r.T$ with probability $r.p$. Thus, all the tuples of a TP relation $\mathbf{r}$ that are valid at time point t with a given probability are included in the *probabilistic snapshot* of $\mathbf{r}$ at t. Specifically, we obtain the probabilistic snapshot of a TP relation $\mathbf{r}$ with schema $R^{\text{Tp}} = (F, \lambda, T, p)$ at time point t by applying the *timeslice operator* τ_t^{p} [PTB18].

The semantics of the TP data model centered around two properties: TP snapshot reducibility and TP change preservation. Snapshot reducibility states that a probabilistic snapshot of the result of an m-ary TP operation $op^{\text{Tp}}(\mathbf{r}_1, \ldots, \mathbf{r}_m)$ at any time point t is equivalent to the result derived from the corresponding probabilistic operation op^{p} on the probabilistic snapshots of the input relations at t. Applying an atemporal operation over all probabilistic snapshots thus is consistent with snapshot reducibility in temporal databases and implies that the result at any time point t, both in terms of probability values and facts, is determined only by the input tuples that are valid at t. The application of op^{p} guarantees that the computations at each time point will yield Boolean lineage expressions that are consistent with the possible-worlds semantics [STW08, BSH$^+$08].

As example for TP snapshot reducibility, consider the query of Fig. 3.1b over the relations of Fig. 3.1a. According to the lineage expression of tuple ('Ann, ZAK, hotel$_1$', [4,6), $a_1 \wedge b_3$, 0.42), at $t = 2$, the fact f = 'Ann, ZAK, hotel$_1$' has been derived from the input tuples a_1 and b_3, i.e., the only input tuples valid at this time point and whose facts could be combined to form f. Since the probability of f at $t = 4$ is only affected by the probabilities of a_1 and b_3, it can be computed based on the lineage expression $a_1 \wedge b_3$.

Intuitively, TP change preservation ensures that only consecutive time points of tuples with equivalent lineage expressions are grouped into intervals. For example, the output tuples ('Ann, ZAK, -', [2,4), a_1, 0.7) and ('Ann, ZAK, -', [4,5), $a_1 \wedge \neg b_3$, 0.42) were not merged into the interval $[2,5)$, since they do not have equivalent lineages. Change preservation guarantees that a fact is valid over the same possible worlds with maximal intervals. As a result, the lineage expression at all time points in the interval of a result tuple is the same and the time points outside this interval have different lineage expressions.

3.4 Negation in TPDBs

The characterization of joins as operations with and without negation has been well established in databases [FO16]. As illustrated in Table 3.1, the Cartesian product and the inner join do not include negation since they record information valid in both input relations. On the contrary, the anti-join is an operation purely based on negation and the result of outer joins is a combination of the above.

Table 3.1: Join Operations Categorized Based on Negation

	Operations
WITHOUT	$\times$, $\bowtie$
WITH	$\triangleright$
MIXED	$\overline{\bowtie}$, $\bowtie\!\!\!\subset$, $\overline{\bowtie\!\!\!\subset}$

Join operations without negation are based on the concept of producing combinations. Given a tuple r of the left relation and a θ-condition, an output tuple is produced for each tuple s of the right relation that satisfies θ and that *overlaps* with r. The attributes of the output tuple based on r and s are defined only based on these two input tuples. In Figure 3.3, the result of the inner join between the TP relations of Fig. 3.1a is illustrated. At time point $t = 5$, tuple a_1 of

the left input relation is combined with tuples b_2 and b_3 under the condition θ : **a**.Loc = **b**.Loc. The corresponding output tuples are ('Ann, ZAK, hotel$_1$', $a_1 \wedge b_2$, [5,8), 0.42) and ('Ann, ZAK, hotel$_2$', $a_1 \wedge b_3$, [4,6), 0.49).

Q = **a** $\bowtie_{\theta}^{\mathrm{Tp}}$ **b**

Name	*Loc*	*Hotel*	λ	T	p
Ann	ZAK	hotel$_1$	$a_1 \wedge b_3$	[4,6)	0.49
Ann	ZAK	hotel$_2$	$a_1 \wedge b_2$	[5,8)	0.42

Figure 3.3: TP inner join on the relations of Fig. 3.1a with θ : **a**.Loc = **b**.Loc.

A join with negation is performed over a positive relation **p** and a negative relation relation **n**. In conventional databases, operations with negation disqualify an input tuple of the positive relation if its attributes match the attributes in a tuple of the negative relation. In temporal databases, the existence of a matching tuple in the negative relation does not disqualify the tuple of **p** itself but time points at which it is valid [BBJ98, BJ09]. In probabilistic databases, where tuples have a probability to be true or false, the existence of a matching tuple in **n** only reduces the probability with which a tuple is included in the output [Suc09, WRS08].

The result of a temporal-probabilistic join with negation includes, at each time point, the probability with which a tuple $\tilde{p}$ of the positive relation **p** matches no tuple in the negative relation **n** under a predicate θ. There are two cases when $\tilde{p}$ matches no tuple of **n**. Firstly, this occurs at time points when either no tuple of **n** has a non-zero probability to be valid or no valid tuple of **n** satisfies the θ-condition. In this case, the $\tilde{p}$ remains *unmatched* and the probability of the output tuple produced is equal to the probability of $\tilde{p}$.

Secondly, the non-existence of a matching tuple for $\tilde{p}$ in **n** occurs when all the valid tuples of **n** that match $\tilde{p}$ are all false. This case relates to the probabilistic dimension where, at each time point, the non-existence of a matching tuple has a probability to occur and thus $\tilde{p}$ is not disqualified for the output. The output fact is determined by a tuple of the positive relation whereas for the computation of the corresponding probability we need to consider the negating form of the probabilities for the matching tuples of the negative relation. In case one of the matching tuples in **n** has probability equal to 1, the output tuple has 0 probability to be *true*.

Example 12. In Fig. 3.4, the TP anti-join between the relations of Fig. 3.1a is illustrated. It contains, at each time point, the probabilities that clients want to visit a location and no hotel is available. Tuple ('Ann, ZAK', a_1, [2,4), 0.7) corresponds to the case when a tuple of the positive relation remains unmatched since there is no hotel in *ZAK* that has a probability to be available

$Q = \mathbf{A} \triangleright^{Tp}_{\theta} \mathbf{B}$

Name	*Loc*	λ	T	p
Ann	ZAK	a_1	[2,4)	0.7
Ann	ZAK	$a_1 \wedge \neg b_3$	[4,5)	0.21
Ann	ZAK	$a_1 \wedge \neg(b_3 \vee b_2)$	[5,6)	0.084
Ann	ZAK	$a_1 \wedge \neg b_2$	[6,8)	0.28
Jim	WEN	a_2	[7,10)	0.8

Figure 3.4: TP anti-join for the relations of Fig. 3.1a with θ : **a**.Loc = **b**.Loc.

in the interval [2,4). Tuple ('Ann, ZAK', $a_1 \wedge \neg(b_3 \vee b_2)$, [5,6), 0.084) corresponds to the case when the the matching tuples of the negative relaton **b** is $false$.

TP outer joins are a combination of joins with and without negation and all of the cases described above should be covered in the result. What differs for outer joins when the temporal and the probabilistic dimension coexist is that two outcomes might arise at a time point. For example, the TP left-join $\mathbf{r} \mathbin{⟕^{Tp}} \mathbf{s}$ includes, at each time point, the facts f for which there is a non-zero probability either to be matched with a tuple in **r** or not to be matched with a tuple in **s** based on a predicate θ. In the TP left-join of Fig. 3.1b, at time point $t = 4$, tuple a_1 is combined with tuple b_3 and forms the output tuples ('Ann, ZAK, hotel_2', $a_1 \wedge b_3$, [4,6), 0.49) and ('Ann, ZAK', -, $a_1 \wedge \neg b_3$), [4,5), 0.21) under the condition that b_3 is $true$ and $false$ respectively. Finally, for the case of the full outer join, the two input relations are both positive and negative at the same time.

3.5 Generalized Windows

In TP set operations, redundant computations that occur when interval and lineage computation are decoupled are avoided by producing output tuples based on *lineage-aware temporal windows*. They are restricted to cases when input tuples with the same fact are combined and when at most one input tuple of each relation is valid over an output interval. In order to overcome these restrictions, we present *generalized lineage-aware temporal windows* and show how, given two TP relations and a θ condition, all windows we produce can be grouped into three disjoint sets: the unmatched, the overlapping and the negating windows. The windows lead to the creation of output tuples for TP joins, matching each of the cases described in section 3.4.

The use of a θ condition in TP outer joins and anti joins requires to pair input tuples that include different facts and to combine multiple input tuples that are valid over an interval and satisfy θ. The ***generalized lineage-aware temporal window*** is a mechanism created based on two TP relations **r** and **s** and has schema (F_r, F_s, T, $\lambda_{\mathtt{r}}$, $\lambda_{\mathtt{s}}$). F_r and F_s are the facts included in tuples of relations **r** and **s** over interval T, respectively. $\lambda_{\mathtt{r}}$ is the disjunction of the lineage expressions of the input tuples of the input relation **r** that are valid over T, include F_r and satisfy θ. $\lambda_{\mathtt{s}}$ is the disjunction of the lineage expressions of the input tuples of the input relation **s** that are valid over T, include F_s and satisfy θ.

The ***generalized lineage-aware temporal windows*** produced based on two TP relations and a θ-condition are grouped based on the case of TP joins with and without negation that they cover. In the remainder of this section, we introduce unmatched, overlapping and negating windows. These three sets are disjoint, i.e. the windows of each set share different characteristics.

Definition 6. *(Unmatched Windows)* Let **r** and **s** be TP relations with schema (F, λ, T, p) and θ a condition between the non-temporal attributes of **r** and **s**. The unmatched windows $\mathbf{W}_{\mathrm{U}}(\mathbf{r};\mathbf{s},\theta)$ of **r** with respect to **s** and θ are defined as follows:

$$
\begin{aligned}
&\tilde{w} \ \in \mathbf{W}_{\mathrm{U}}(\mathbf{r};\mathbf{s},\theta) \Longleftrightarrow \\
&\tilde{w}.\lambda_s = \mathtt{null} \ \wedge \tilde{w}.F_s = \mathtt{null} \ \wedge \\
&\forall t \in \tilde{w}.T\,(\exists r \in \mathbf{r}\ (\tilde{w}.F_r = r.F \wedge \tilde{w}.\lambda_r \equiv r.\lambda) \wedge \lambda_t^{\mathbf{s},\theta_{\tilde{w}}} = \mathtt{null}) \wedge \\
&\forall t' \notin \tilde{w}.T\,(\nexists r \in \mathbf{r}(\tilde{w}.F_r = r.F \wedge \tilde{w}.\lambda_r \equiv r.\lambda) \vee \tilde{w}.\lambda_s \not\equiv \lambda_{t'}^{\mathbf{s},\theta_{\tilde{w}}})
\end{aligned}
$$

The *unmatched windows* $\mathbf{W}_{\mathrm{U}}(\mathbf{r};\mathbf{s},\theta)$ correspond to output tuples that cover the case of a tuple of **r** not matching a tuple of **s**, described in Section 3.4. A window w in $\mathbf{W}_{\mathrm{U}}(\mathbf{r};\mathbf{s},\theta)$ is an unmatched window and spans over the interval or a subinterval of a tuple r of **r** during which all tuples of **s** are either not valid or don't match θ. The fact F_r and the lineage λ_r of w are determined by r while F_s and λ_s are set to `null`. The interval of the window w corresponds to a maximal subinterval of r. In other words, as stated in the 3^{rd} line of the definition, at every time point outside the $\tilde{w}.T$, either tuple r stops being valid or a tuple of **s** starts being valid.

Example 13. In Fig. 3.5, the TP relations **a** and **b** of Fig. 3.1 are illustrated along with the corresponding unmatched windows. Different colors are used to annotate different facts: black is used for 'Ann, ZAK', red for 'John, WEN', green for 'hotel$_3$, SOR', yellow for 'hotel$_2$, ZAK', and blue for 'hotel$_1$, ZAK'. Wavy lines are used for tuples of an input relation that match no

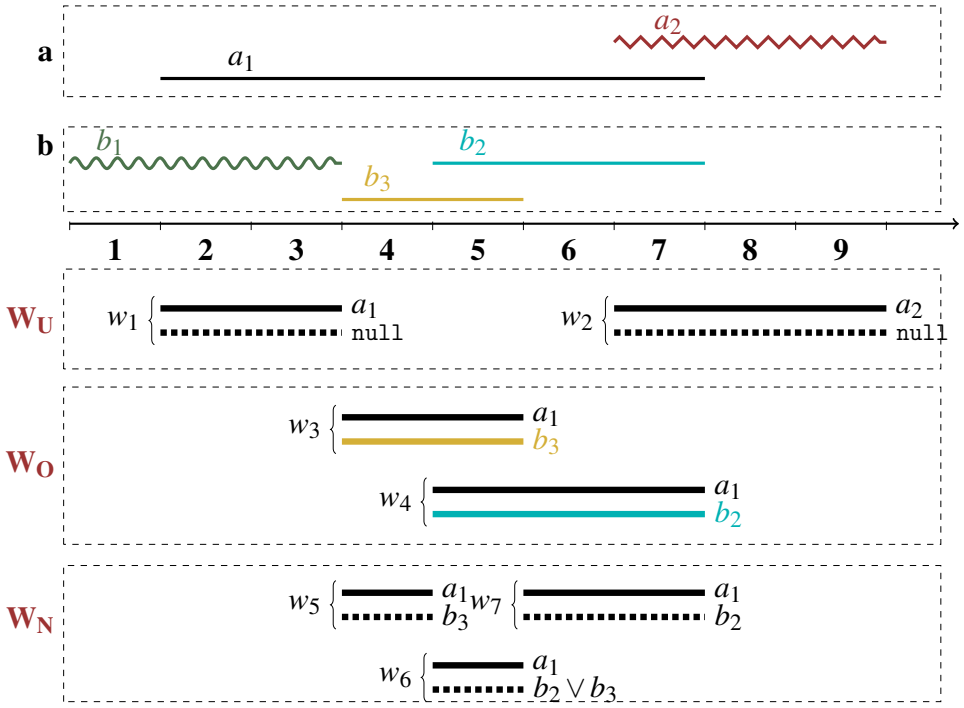


Figure 3.5: All windows of **a** with respect to **b** with θ : **a**.Loc = **b**.Loc

tuple of the other relation for θ. The window w_1 = ('Ann, ZAK, `null`', [2, 4), a_1, `null`) belongs to the unmatched windows $\mathbf{W}_{\text{U}}(\mathbf{a};\mathbf{b},a.Loc = b.Loc)$ of relation **a** with respect to relation **b** and predicate $\theta : a.Loc = b.Loc$. A straight line in black color indicates that the fact $w_1.F_r$ = 'Ann, ZAK' and the lineage $w_1.\lambda_r = a_1$ match the corresponding attributes of tuple a_1. A dotted line indicates that the fact $w_1.F_s$ is set to `null` and so is $w_1.\lambda_s$. At $t = 4$, a_1 is still valid whereas $\lambda_4^{\mathbf{b},\theta_{w_1}} = b_3$, indicating that a tuple of **b** starts being valid at $t = 4$ and thus the interval $[2,4)$ is maximal.

Definition 7. *(Overlapping Windows)* Let **r** and **s** be TP relations with schema (F, λ, T, p) and θ a condition between the non-temporal attributes of **r** and **s**. The *overlapping windows* $\mathbf{W}_{\text{O}}(\mathbf{r};\mathbf{s},\theta)$ of **r** with respect to **s** and θ are defined as follows:

$$\tilde{w} \in \mathbf{W}_{\text{O}}(\mathbf{r};\mathbf{s},\theta) \Longleftrightarrow$$
$$\exists r \in \mathbf{r},\ s \in \mathbf{s}\ (\ \tilde{w}.F_r = r.F\ \wedge\ \tilde{w}.F_s = s.F\ \wedge\ \theta\ \wedge$$
$$\tilde{w}.\lambda_r \equiv r.\lambda\ \wedge\ \tilde{w}.\lambda_s \equiv s.\lambda\ \wedge\ \tilde{w}.T = r.T \cap s.T\ )$$

The overlapping windows cover the case of output tuples for TP joins without negation, also necessary for outer joins. A window w in $\mathbf{W}_{\mathrm{O}}(\mathbf{r};\mathbf{s},\theta)$ spans a maximal interval over which a tuple r of $\mathbf{r}$ overlaps with a tuple s from $\mathbf{s}$ and the predicate θ is satisfied. Tuple r includes the fact F_r and has lineage λ_r while F_s and λ_s correspond to the fact and lineage of tuple s. The interval of the window that is produced by the pair of tuples r and s corresponds to the overlap of the intervals of r and s ($\tilde{w}.T = r.T \cap s.T$). For example, in Fig. 3.5, the window w_3 = ('Ann, ZAK', 'hotel$_1$', [4,6), a_1, b_3) is an overlapping window $\mathbf{W}_{\mathrm{O}}(\mathbf{a};\mathbf{b},a.Loc = b.Loc)$. The window w_3 consists of a blue and a black straight line, indicating that facts F_r and F_s of w_3 correspond to the facts of tuples a_1 and b_3. These two tuples overlap and match according to the condition $\theta : a.Loc = b.Loc$.

Proposition 2. *Let* $\mathbf{r}$ *and* $\mathbf{s}$ *be temporal-probabilistic relations with schema (F, λ, T, p) and* θ *a condition between the attributes of* $\mathbf{r}$ *and* $\mathbf{s}$*. For the sets of overlapping windows of the two relations with respect to one another it holds:*

$$\mathbf{W}_{O}(\mathbf{r};\mathbf{s},\theta) = \pi_{F_s,\, F_r,\, T,\, \lambda_s,\, \lambda_r}(\mathbf{W}_{O}(\mathbf{s};\mathbf{r},\theta))$$

The symmetricity of overlapping windows guarantees that, from the overlapping windows of two TP relations, regardless of which relation we have used as reference, we can derive the input tuples valid over this interval. As a result, the computation of either $\mathbf{W}_{\mathrm{O}}(\mathbf{r};\mathbf{s},\theta)$ or $\mathbf{W}_{\mathrm{O}}(\mathbf{s};\mathbf{r},\theta)$ suffices. This characteristic provides a leverage over related approaches that adjust the intervals of each input relation with respect to the other but need to perform equality checks to be able to characterize an interval as an overlapping one, let alone record the lineages of the tuples valid over this interval [DBG12, DBGJ16b, KMV$^+$13].

Definition 8. *(Negating Windows)* Let $\mathbf{r}$ and $\mathbf{s}$ be TP relations with schema (F, λ, T, p) and θ a condition between the attributes of $\mathbf{r}$ and $\mathbf{s}$. The negating windows $\mathbf{W}_{\mathrm{N}}(\mathbf{r};\mathbf{s},\theta)$ of $\mathbf{r}$ with respect to $\mathbf{s}$ and θ are defined as follows:

$$\begin{aligned}
&\tilde{w} \in \mathbf{W}_{\mathrm{N}}(\mathbf{r},\mathbf{s},\theta) \Longleftrightarrow \\
&\forall t \in \tilde{w}.T\ (\exists r \in \mathbf{r}\ (\tilde{w}.F_r = r.F\ \wedge\ \tilde{w}.\lambda_r \equiv r.\lambda) \wedge \\
&\qquad \tilde{w}.F_s = \texttt{null}\ \wedge\ \lambda_t^{\mathbf{s},\theta_{\tilde{w}}}\ \neq \texttt{null}\ \wedge \tilde{w}.\lambda_s = \lambda_t^{\mathbf{s},\theta_{\tilde{w}}}\ ) \wedge \\
&\forall t' \notin \tilde{w}.T\ (\ \nexists r \in \mathbf{r}\ (\tilde{w}.F_r = r.F\ \wedge\ \tilde{w}.\lambda_r \equiv r.\lambda)\ \vee \tilde{w}.\lambda_s \not\equiv \lambda_{t'}^{\mathbf{s},\theta_{\tilde{w}}}\ )
\end{aligned}$$

The negating windows $\mathbf{W}_{\text{N}}(\mathbf{r},\mathbf{s},\theta)$ of the TP relation **r** with respect to the TP relation **s** includes windows during which a fact is included in a tuple r of **r** as well as in multiple tuples of **s** that are valid and satisfy the θ-condition. Negating windows are suitable for producing output tuples where all the tuples of **s** that match under θ a tuple r of **r** including the fact F_r are $false$, as described in Section 3.4. Thus, the fact F_r and the lineage λ_r of the window are determined by r, F_s is set to `null` and λ_s is the disjunction of the lineages of all the tuples in **s** that are matched with r.

In Fig. 3.5, the window w_6 = ('Ann, ZAK', `null`, $[5,6)$, a_1, $b_3 \vee b_2$) is a negating window. The black straight line in w_6 indicates that its fact F_r and its lineage λ_r correspond to the lineage and fact of a_1. The fact F_s is `null`, illustrated by a dotted line. The lineage λ_s equals the disjunction of the tuples b_2 and b_3 that satisfy θ over the interval $[5,6)$. The interval $[5,6)$ is a maximal interval since for every other time point either a_1 is not valid or different tuples of **b** satisfy θ. For example, at $t=5$, $\lambda_5^{\mathbf{b},'ZAK'=b.Loc} = b_3 \vee b_2$ while at $t=6$, $\lambda_6^{\mathbf{b},'ZAK'=b.Loc} = b_2$.

3.6 Relational Algebra Definitions

In this section, we express the results of TP outer-joins and TP anti-join using the generalized lineage-aware temporal windows produced for two input relations and a predicate. Each generalized lineage-aware temporal window $w = (F_r, F_s, T, \lambda_r, \lambda_s)$ corresponds to exactly one output tuple. This tuple is formed using the facts (F_r, F_s) and interval T in their exact form while the output lineage is formed by combining λ_r and λ_s with the proper lineage-concatenation function. Each set of windows is matched with a unique function of Table 3.3. For *overlapping windows* we use the function ***and***, for *negating windows* we use ***andNot*** and for *unmatched windows* only λ_r is passed on to the output lineage.

Since each window covers a maximal interval, we guarantee that each operation satisfies TP change preservation. Using the appropriate lineage-concatenation functions for each window, we guarantee that the output lineage expressions comply with TP snapshot reducibility. The same holds for the output facts since they are the same for each time point of a window interval and thus of an output interval. For every join with negation we include all windows of the positive relation with respect to the negative one. The full outer-join is symmetric. In this case, the input relations are perceived as both positive and thus, the unmatched and negating windows of each

relation with respect to the other are computed. Due to the symmetricity of the overlapping windows (Prop. 2), the overlapping windows only need to be computed once.

Definition 9. *(TP Joins) Let* $\mathbf{r}$ *and* $\mathbf{s}$ *be temporal-probabilistic relations with schema (F, λ, T, p). Given the lineage-concatenation functions depicted in Table 3.3, we define the join operations* $\mathbf{r} \rtimes^{Tp}_{\theta} \mathbf{s}$, $\mathbf{r} \bowtie^{Tp}_{\theta} \mathbf{s}$, $\mathbf{r} \triangleright^{Tp}_{\theta} \mathbf{s}$ *and* $\mathbf{r} \rtimes\!\bowtie^{Tp} \mathbf{s}$ *as follows:*

$\tilde{r} \in \mathbf{r} \rtimes^{\text{Tp}}_{\theta} \mathbf{s}$	$\iff$	$\pi_{F_r,\, F_s,\, \text{T},\, \lambda_r/\lambda}(\mathbf{W}_{\text{U}}(\mathbf{r};\mathbf{s},\theta)) \cup$ $\pi_{F_r,\, F_s,\, \text{T},\, \text{and}(\lambda_r,\lambda_s)/\lambda}(\mathbf{W}_{\text{O}}(\mathbf{r};\mathbf{s},\theta)) \cup$ $\pi_{F_r,\, F_s,\, \text{T},\, \text{andNot}(\lambda_r,\lambda_s)/\lambda}(\mathbf{W}_{\text{N}}(\mathbf{r};\mathbf{s},\theta))$
$\tilde{r} \in \mathbf{r} \bowtie^{\text{Tp}}_{\theta} \mathbf{s}$	$\iff$	$\pi_{F_r,\, F_s,\, \text{T},\, \lambda_r/\lambda}(\mathbf{W}_{\text{U}}(\mathbf{s};\mathbf{r},\theta)) \cup$ $\pi_{F_r,\, F_s,\, \text{T},\, \text{and}(\lambda_r,\lambda_s)/\lambda}(\mathbf{W}_{\text{O}}(\mathbf{s};\mathbf{r},\theta)) \cup$ $\pi_{F_r,\, F_s,\, \text{T},\, \text{andNot}(\lambda_r,\lambda_s)/\lambda}(\mathbf{W}_{\text{N}}(\mathbf{s};\mathbf{r},\theta))$
$\tilde{r} \in \mathbf{r} \triangleright^{\text{Tp}}_{\theta} \mathbf{s}$	$\iff$	$\pi_{F_r,\, \text{T},\, \lambda_r/\lambda}(\mathbf{W}_{\text{U}}(\mathbf{r};\mathbf{s},\theta)) \cup$ $\pi_{F_r,\, \text{T},\, \text{andNot}(\lambda_r,\lambda_s)/\lambda}(\mathbf{W}_{\text{N}}(\mathbf{r};\mathbf{s},\theta))$
$\tilde{r} \in \mathbf{r} \rtimes\!\bowtie^{\text{Tp}}_{\theta} \mathbf{s}$	$\iff$	$\pi_{F_r,\, F_s,\, \text{T},\, \lambda_r/\lambda}(\mathbf{W}_{\text{U}}(\mathbf{r};\mathbf{s},\theta)) \cup$ $\pi_{F_r,\, F_s,\, \text{T},\, \text{and}(\lambda_r,\lambda_s)/\lambda}(\mathbf{W}_{\text{O}}(\mathbf{r};\mathbf{s},\theta)) \cup$ $\pi_{F_r,\, F_s,\, \text{T},\, \text{andNot}(\lambda_r,\lambda_s)/\lambda}(\mathbf{W}_{\text{N}}(\mathbf{r};\mathbf{s},\theta)) \cup$ $\pi_{F_s,\, F_r,\, \text{T},\, \lambda_r/\lambda}(\mathbf{W}_{\text{U}}(\mathbf{s};\mathbf{r},\theta)) \cup$ $\pi_{F_s,\, F_r,\, \text{T},\, \text{andNot}(\lambda_r,\lambda_s)/\lambda}(\mathbf{W}_{\text{N}}(\mathbf{s};\mathbf{r},\theta))$

Table 3.2: Definition of TP tuple-based operations

$\textbf{\textit{and}}(\lambda_1,\lambda_2)$	$= (\lambda_1) \wedge (\lambda_2)$	
$\textbf{\textit{andNot}}(\lambda_1,\lambda_2)$	$= \begin{cases} (\lambda_1) & \text{if } \lambda_2 = \texttt{null} \\ (\lambda_1) \wedge \neg(\lambda_2) & \text{otherwise} \end{cases}$	
$\textbf{\textit{or}}(\lambda_1,\lambda_2)$	$= \begin{cases} (\lambda_1) & \text{if } \lambda_2 = \texttt{null} \\ (\lambda_2) & \text{if } \lambda_1 = \texttt{null} \\ (\lambda_1) \vee (\lambda_2) & \text{otherwise} \end{cases}$	

Table 3.3: Definition of lineage-concatenation functions.

Example 14. In the TP anti-join in Figure 3.4, the unmatched and the negating windows of Fig. 3.5 are used. The unmatched window ('Ann, ZAK', `null`, [2,4), a_1, `null`) is transformed to the output tuple ('Ann, ZAK', -, [2,4), a_1) and the negating window ('Ann, ZAK', `null`, [5,6), a_1, $b_3 \vee b_2$) is transformed to the output tuple ('Ann, ZAK', [5,6), $a_1 \wedge \neg(b_3 \vee b_2)$).

3.7 Algorithms

In this section, we introduce algorithms to compute all *generalized lineage-aware temporal windows* needed for the result of TP outer-joins and TP anti-join. The overlapping windows is the basis for the computation of unmatched and negating windows for two TP relations **r** and **s**. In order to produce the unmatched windows of **r** with respect to **s**, we identify subintervals of **r** during which there is no overlap or match with a tuple of **s**, i.e., subintervals that do not correspond to any overlapping window. Similarly, each of the negating windows of **r** with respect to **s** spans a subinterval where all the tuples of **s** that overlap and match with a tuple r of **r** are false and thus lineage information from all the overlapping windows that are valid over this subinterval and involving r must be combined.

Our approach leverages conventional outer joins with predicates and sweeping-window [PTB18] algorithms to compute overlapping, unmatched and negating windows, respectively. $LAWA_{\text{U}}$ (Algorithm 5) produces the unmatched windows based on the overlapping ones. $LAWA_{\text{N}}$ (Algorithm 6) is applied on the result of $LAWA_{\text{U}}$, passing the umatched and overlapping windows directly to the output while simultaneously recording the information of the overlapping windows that are needed to produce each negative window. In contrast to existing sweeping algorithms, $LAWA_{\text{U}}$ and $LAWA_{\text{N}}$ are applied on windows instead of tuples. They do not only update structures that combine input information to produce output windows but they also copy input windows to the output. They are operating in an incremental manner, thus avoiding recomputing the overlapping windows multiple times.

3.7.1 Overlapping Windows

The attributes of every overlapping window is based on a pair of tuples, one from each input relation, valid over the window. The computation of overlapping windows is performed using the conventional outer join $\mathbf{r} \overline{\bowtie} \mathbf{s}$ of the input relations **r** and **s** with schema (F, λ, T, p), using the overlapping predicate $\theta_o : r.T \cap s.T$ and a θ-condition on the non-temporal attributes, as provided in the TP join to be computed. The result of $\mathbf{r} \overline{\bowtie}_{\theta_o \wedge \theta} \mathbf{s}$ contains windows of relation **r** with respect to relation **s** and condition θ, enhanced with the time-interval of the tuple of r valid over each window, and has schema: (F_r, λ_r, F_s, λ_s, $[O_s, O_e)$, $[T_s, T_e)$). (F_r, $[T_s, T_e)$, λ_r) correspond to the fact, interval and lineage of a tuple r in **r**. (F_s, λ_s) correspond to the fact and lineage of a tuple s in **s**. $[O_s, O_e)$ is the interval during which the two tuples r and s overlap.

The tuples of the join $\mathbf{r} ⟕_{\theta_o \wedge \theta} \mathbf{s}$ for which all attributes are not `null` constitute the set of overlapping windows $\mathbf{W}_{\text{o}}(\mathbf{r};\mathbf{s},\theta)$. The interval of these windows is equal to $[O_s, O_e)$. The use of the conventional left join results in pairs with `null` attributes which correspond to tuples of $\mathbf{r}$ not combined with any tuple of $\mathbf{s}$, i.e., to unmatched windows.

	X					
	F_r	λ_r	F_s	λ_s	$[O_s, O_e)$	$[T_s, T_e)$
$\mathbf{x_1}$	'Ann, ZAK'	a_1	'hotel$_1$, ZAK'	b_3	[4,6)	[2,8)
$\mathbf{x_2}$	'Ann, ZAK'	a_1	'hotel$_2$, ZAK'	b_2	[5,8)	[2,8)
$\mathbf{x_3}$	'Jim, WEN'	a_2	`null`	`null`	`null`	[9,12)

Figure 3.6: The result of $\mathbf{a} ⟕_{\text{r.T} \cap \text{s.T} \wedge \mathbf{a}.\text{Loc}=\mathbf{b}.\text{Loc}} \mathbf{b}$.

3.7.2 Unmatched Windows

The unmatched windows of a TP relation $\mathbf{r}$ with respect to a TP relation $\mathbf{s}$ and a condition θ are computed in two phases. Firstly, the result of the conventional outer join $\mathbf{r} ⟕_{\theta_o \wedge \theta} \mathbf{s}$, that computes the overlapping windows, also contains windows for which (F_s, λ_s) as well as $[O_s, O_e)$ are set to `null`. These windows are a subset of the unmatched windows $\mathbf{W}_{\text{U}}(\mathbf{r};\mathbf{s},\theta)$. They are created based on input tuples of $\mathbf{r}$ that don't overlap or don't match under θ with any tuple in $\mathbf{s}$. The interval of each such window is equal to the interval $[T_s, T_e)$ of the tuple of $\mathbf{r}$ valid.

Secondly, the algorithm *LAWA*$_{\text{U}}$ (Lineage-Aware Window Advancer for Unmatched Windows) extends the result $\mathbf{X}$ of the join with the remaining unmatched windows, i.e., the windows that span only a *subinterval* of a tuple in $\mathbf{r}$ during which no tuple in $\mathbf{s}$ is valid or satisfies θ. For these unmatched windows to be created, the windows already in the result $\mathbf{X}$ are grouped according to the fact F_r and the interval $[T_s^r, T_e^r)$ of the tuple in $\mathbf{r}$ to which they correspond. Within each group, the tuples are sorted on the starting point of their overlapping intervals (Os) and the order of tuples with equal starting points does not matter. The algorithm performs a sweep of the interval $[T_s, T_e)$ of each tuple of $\mathbf{r}$. It copies the overlapping windows ($[Os, Oe) \neq$ `null`) relating to r to the output. At the sae time, given the subintervals the overlapping windows span and the initial interval $[T_s, T_e)$ of the tuple of $\mathbf{r}$, it identifies the subintervals during which there is no overlap with a tuple in $\mathbf{s}$, i.e., no overlapping window, and uses them to produce the remaining unmatched windows.

Algorithm 5: LAWA$_{\text{U}}$(status)

```
1  (prevWindTe, currFact_r, currFact_s, λ_r, wind) = status;
2  if wind = null then return null;
3  do
4      if prevWindTe = −1 then
5          windTs = wind.T_s;
6          currFact_r = wind.F_r; λ_r = wind.λ_r;
7      else windTs = prevWindTe;
8      λ_s = null; currFact_s = null;
9      if wind.O_s = windTs then
10         λ_s = wind.λ_s; currFact_s = wind.F_s;
11     if λ_s ≠ null then windTe = wind.O_e ;
12     else if windTs = wind.T_s ∧ wind.O_s ≠ null then
13         windTe = wind.O_s;
14     else if wind.O_s = null ∨ windTs = wind.O_e then
15         next = getNextOf(wind );
16         if next ≠ null ∧ F_r = next.F_r then windTe = next.O_s;
17         else windTe = wind.T_e;
18         wind = next ;
19     if windTe = wind.T_e then prevWindTe = −1;
20     else prevWindTe = windTe;
21 while windTs ≥ windTe;
22 window = (currFact, windTs, windTe, λ_r , λ_s) ;
23 status = (prevWindTe, currFact_r, currFact_s, λ_r, wind);
24 return (window, status);
```

The input of $LAWA_{\text{U}}$ is a context node ($\texttt{status}_U$) with information on the status of the algorithm: the right boundary of the last output window (prevWindTe), ($\texttt{currFact}_\texttt{r}$) and ($\texttt{currFact}_\texttt{s}$) are respectively the facts of the tuple of **r** and the tuple of **s** that are valid over the sweeping window $[\texttt{windTs}, \texttt{windTe})$ and wind is the next window of **X** to be processed. Given that all the necessary information is recorded, at each call a generalized lineage-aware temporal window (Line 22) is returned as well as the $\texttt{status}_\text{U}$ necessary for the next call. In the first call (firstCall) of the algorithm, the first window of **X** is fetched, rValid and sValid are initialized to null and prevWindTe is initialized to -1.

Initially, the left boundary windTs of the new window as well as the fact and the lineage of the valid tuple of **r** are determined. If a new group is being processed, i.e., if prevWindTe is equal to -1, windTs is determined by the starting point of the first window *wind* of the new group

(Lines 4-6). In this case, the fact $\texttt{currFact}_\texttt{r}$ and the lineage λ_r of the valid tuple of **r** are also extracted from *wind*. If the processing of a group continues, the interval of the new window will be adjacent to the previous one, with `windTs` = `prevWindTe` (Line 7) while $\texttt{currFact}_\texttt{r}$ and λ_r remain unchanged.

In order to determine the fact and the lineage of the tuple of **s** valid over the output window, we need to check the starting point `windTs` of the window matches the starting point O_s of an overlapping window in **X**. If satisfied, this condition (Line 9) indicates that there is a tuple of **s** valid over the window and thus the fact $\texttt{currFact}_\texttt{s}$ and lineage λ_s are determined based on the corresponding attributes of `wind`. Otherwise, they are set to `null`.

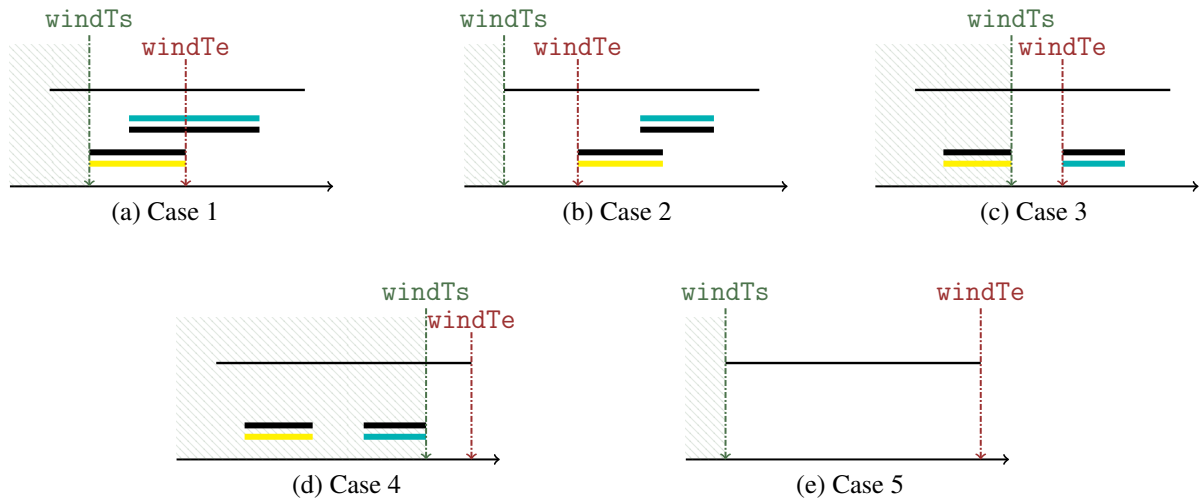


Figure 3.7: Cases for determining *windTe* in LAWA$_U$ Algorithm. The windows in **X** are illustrated as well as the interval of the tuple of the left relation corresponding to the group of windows scanned.

The right boundary `windTe` of the output window is determined based on whether the window is an overlapping or an unmatched one. All the cases covered in Lines 11-18 are annotated in the algorithm and illustrated in Figure 3.7. If the output window is an overlapping window (Case 1), i.e., $\lambda_s \neq \texttt{null}$, its interval corresponds to the overlapping interval in *wind* and thus, `windTe` is set to `wind.Oe`. If the output window is an unmatched window, three different cases are considered based on the position of *windTs* with respect to $[wind.O_s, wind.O_e)$. If the starting point *windTs* coincides with the starting point of the valid tuple of **r** ($\texttt{windTs} = \texttt{wind}.T_s$) and the starting point of an overlapping window *wind* succeds (Case 2), `windTe` is set to the starting point of *wind*. If the starting point of the output window coincides with the ending point of the

overlapping window *wind* (Case 3), the upcoming window *next* is fetched. If *next* is a window of the same group as *wind*, the output window is positioned between two overlapping windows and thus $\texttt{windTe} = \texttt{next}.Os$. However, if *next* belongs to a new group, the current window is positioned at the end of the interval of a valid tuple of **r** (Case 4). Thus $\texttt{windTe} = \texttt{wind}.Te$ and the sweeping progresses to window *next*. The same assignment takes place if *wind* is not an overlapping window but one of the unmatched windows produced by the conventional left outer join (Case 5).

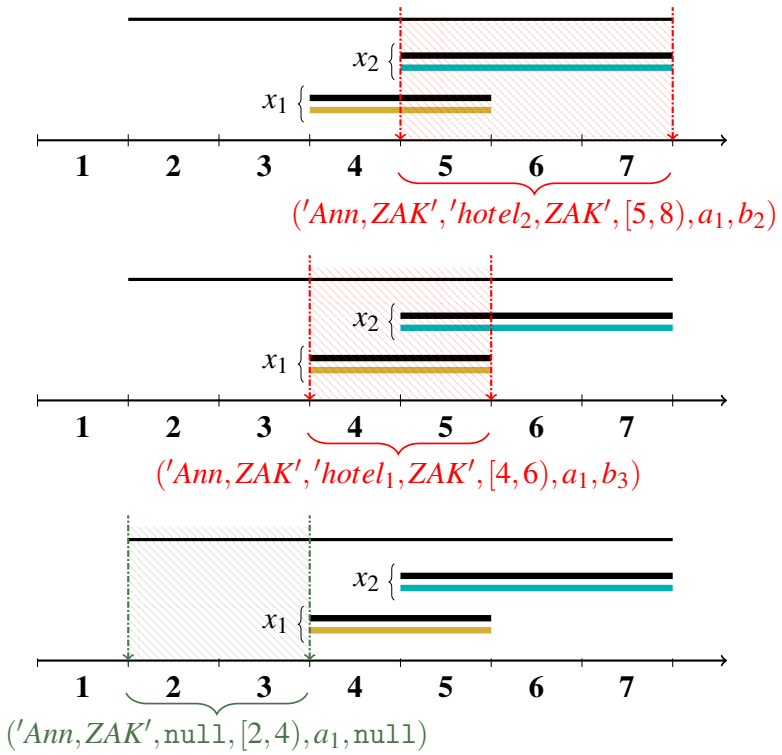


Figure 3.8: LAWA$_U$ on the group with $\texttt{F}_\texttt{r} = 'Ann, ZAK'$ and $\lambda_\texttt{r} = a_1$.

Example 15. In Fig. 3.8, we illustrate two calls of LAWA$_U$ when applied on the relation **X** of Fig 3.6 and more specifically on the group of windows the fact $\texttt{F}_\texttt{r}$ ='Ann, ZAK'. The single blank line corresponds to tuple a_1, the tuple of the left relation **a** valid over all the windows of the group. The window $wind = x_1$ is the first window to be processed. In the first call of LAWA$_U$, illustrated at the bottom of the figure, the processing of a new group starts and $\texttt{windTs}$, $\texttt{currFact}_\texttt{r}$ and $\lambda_\texttt{r}$ are initialized to the starting point, fact and lineage of a_1, respectively. No overlapping window of the same group starts at $\texttt{windTs} = 2$ and thus $\texttt{currFact}_\texttt{s}$ and $\lambda_\texttt{s}$ are set to $\texttt{null}$. According to Case 2, $\texttt{windTe}$ is set to the starting point $wind.O_s$ of the overlapping interval of *wind*. In

the second call of LAWA, the left boundary of the next window to be examined is equal to the right boundary of the previous window, i.e., `windTs` $= 4$, given that the fact ('Ann, ZAK') is still being processed. Currently, `windTs` equals the starting point of the overlapping window x_1 and the facts, lineages and intervals of the output window are fetched from x_1. In the third call, the starting point of the output window is again equal to the starting point of an overlapping window, so the same procedure is followed.

3.7.3 Negating Windows

A negating window of a TP relation **r** with respect to a TP relation **s** and a condition θ spans over intervals where a tuple of **r** could overlap with multiple tuples of **s**. Instead of recomputing the tuples of the input relations that match θ and overlap, we introduce the algorithm $LAWA_{\text{N}}$ (Lineage-Aware Window Advancer for Negating Windows), an algorithm that computes the negating windows based on the overlapping ones. $LAWA_{\text{N}}$ extends the result of $LAWA_{\text{U}}$, already including all the unmatched and overlapping windows, with the negating windows. Given the way windows are produced in $LAWA_{\text{U}}$, its output **Y** consists of windows ordered by the fact of **r** (F_r) as well as by their starting point (Ts). In Fig. 3.9, we have included the result of $LAWA_{\text{U}}$ based on the relations of Fig. 3.1a. $LAWA_{\text{N}}$ sweeps over **Y** and copies all the unmatched and overlapping windows to the output. When a group of overlapping windows with the same fact F_r is encountered, negating windows are created. The intervals of these windows are subintervals of the group of overlapping windows.

Y

	F_r	F_s	λ_r	λ_s	$T = [T_s, T_e)$
y1	'Ann, ZAK'	`null`	a_1	`null`	[2,4)
y2	'Ann, ZAK'	'hotel$_1$, ZAK'	a_1	b_3	[4,6)
y3	'Ann, ZAK'	'hotel$_1$, ZAK'	a_1	b_2	[5,8)
y4	'Jim, WEN'	`null`	a_2	`null`	[9,12)

Figure 3.9: The input of LAWA$_N$

The execution of $LAWA_{\text{N}}$ is based on a context node (`status`$_N$) with information on the status of the algorithm, different than the node used in $LAWA_{\text{U}}$: the right boundary of the last negating window (`prevWindTe`), the fact (`currFact`$_\text{r}$) and the lineage (λ_r) of the tuple of the left relation that is valid over the sweeping window [`windTs`, `windTe`), the next window (`wind`) of **Y** to be processed, the tag `neg` indicating if a negating window will be produced and a priority queue `PQ`.

The priority queue PQ includes (t, λ) pairs indicating the time point t after which the tuple of the right relation with lineage λ stops being valid.

Algorithm 6: LAWA$_N$(status)

```
1  (prevWindTe, F_r, λ_r, PQ, neg, wind) = status;
2  if wind = null ∧ isPQempty() then return (null, null);
3  if firstCall then
4  |  PQ = initializePQ(); prevWindTe = −1; neg = false;
5  if prevWindTe = −1 ∧ wind.λ_r ≠ null then
6  |  F_r = wind.F_r; λ_r = wind.λ_r; prevWindTe = wind.T_s;
7  while out = null do
8  |  if neg = false then
9  |  |  out = wind ;
10 |  |  if wind.F_s = null then wind = getNextTuple();
11 |  |  else
12 |  |  |  neg = true ;
13 |  |  |  addToPQ(wind.T_e, wind.λ_s) ;
14 |  else if wind.F_r = F_r ∧ wind.Ts ≤ prevWindTe then
15 |  |  wind = getNextTuple() ;
16 |  if out = null ∧ wind.F_r = F then
17 |  |  if wind.T_s > prevWindTe then
18 |  |  |  windTe = tForTopOfPQ();
19 |  |  |  if wind.T_s < windTe then
20 |  |  |  |  windTe = wind.T_s;
21 |  |  |  |  λ_s = disjunctLineages(windTe);
22 |  |  |  |  out = (F_r, −, [prevWindTe, windTe), λ_r, λ_s);
23 |  |  |  |  prevWindTe = windTe;
24 |  |  |  |  neg = false;
25 |  |  else if wind.T_s = prevWindTe then neg = false ;
26 |  else if out = null ∧ (¬ isPQempty()) then
27 |  |  windTe = tForTopOfPQ(); λ_s = disjunctLineages(windTe);
28 |  |  out = (F_r, −, [prevWindTe, windTe), λ_r, λ_s);
29 |  |  prevWindTe = windTe; removeTopOfPQ();
30 if isPQempty() then
31 |  prevWindTe = −1; neg = false;
32 status = (prevWindTe, F_r, λ_r, PQ, neg, wind);
33 return (out, status);
```

In the first call of the algorithm (`firstCall`), the first tuple of **Y** is fetched, the priority queue `PQ` is initialized (pointer to `null`), `prevWindTe` is set to -1 and `neg` to $false$. Since negating windows are created based on the overlapping windows, whenever a group of overlapping windows with the same F_r starts, the output fact $\mathtt{F_r}$, the output lineage $\lambda_\mathtt{r}$ and the starting point `prevWindTe` of the output windows are updated to the values of the first tuple of this group for F_r, $\lambda_\mathtt{r}$ and T_s respectively.

$LAWA_{\mathrm{N}}$ outputs an unmatched, overlapping or negating window according to `neg`. When `neg` is $false$ (Line 8), the unmatched or overlapping window to which `wind` points is copied to the output as is (Line 9). If `wind` corresponds to an unmatched window (`wind`.F_s = `null`), we proceed to the next window. However, if it corresponds to an overlapping window, the creation of a negating window is bound to follow and `neg` is set to $true$ (Line 12). In this case, the lineages and ending points of the tuples of the right relation that are valid during the window need to be recorded (Line 13). Thus, we add to the priority queue `PQ` the pair ($wind.T_e$, $wind.\lambda_s$), consisting of the ending point T_e and the lineage λ_s of the valid tuple in the relation **s** as recorded in `wind`.

When `neg` is $true$, the creation of a negating window follows. The starting point of this output window has been determined and is equal to *prevWindTe*. If this time point corresponds to the starting point of the current input window F, the next window needs to be fetched (Line 15) for two reasons. Firstly, if the next window of **Y** is an overlapping window of the same group and starts at *prevWindTe*, the lineage of the tuple of relation **s** valid over this input window needs to be considered for $\lambda_\mathtt{s}$. Secondly, if the next window belongs to the same group, its starting point should be considered as a potential ending point of the output window.

In Lines 16-29, the output negating window is finalized by determining its ending point `windTe` and lineage $\lambda_\mathtt{s}$. The lineage is always determined by disjuncting the lineage expressions of the pairs (t, λ) in the priority queue with t smaller than `windTe`. Thus, $\lambda_\mathtt{s}$ will correspond to the disjuction of the tuples of the negative relation valid over the output interval $[\mathtt{prevWindTe}, \mathtt{windTe})$.

To determine the ending point `windTe` of the window, two cases are considered based on whether the upcoming window *wind* of **Y** includes the same fact as the fact $\mathtt{F_r}$ of the output window. If this is the case, the ending point of the output window is the minimum between the time point of the top pair in the queue, i.e. the smallest ending point of valid tuples in relation **s** that are valid, and the starting point of the upcoming window of **Y**. Therefore, a window is created when there is a change in the tuples of the reference relation that are valid either because a tuple ends or a new tuple begins. After the output negating window is formed, the starting point *prevWindTe* of

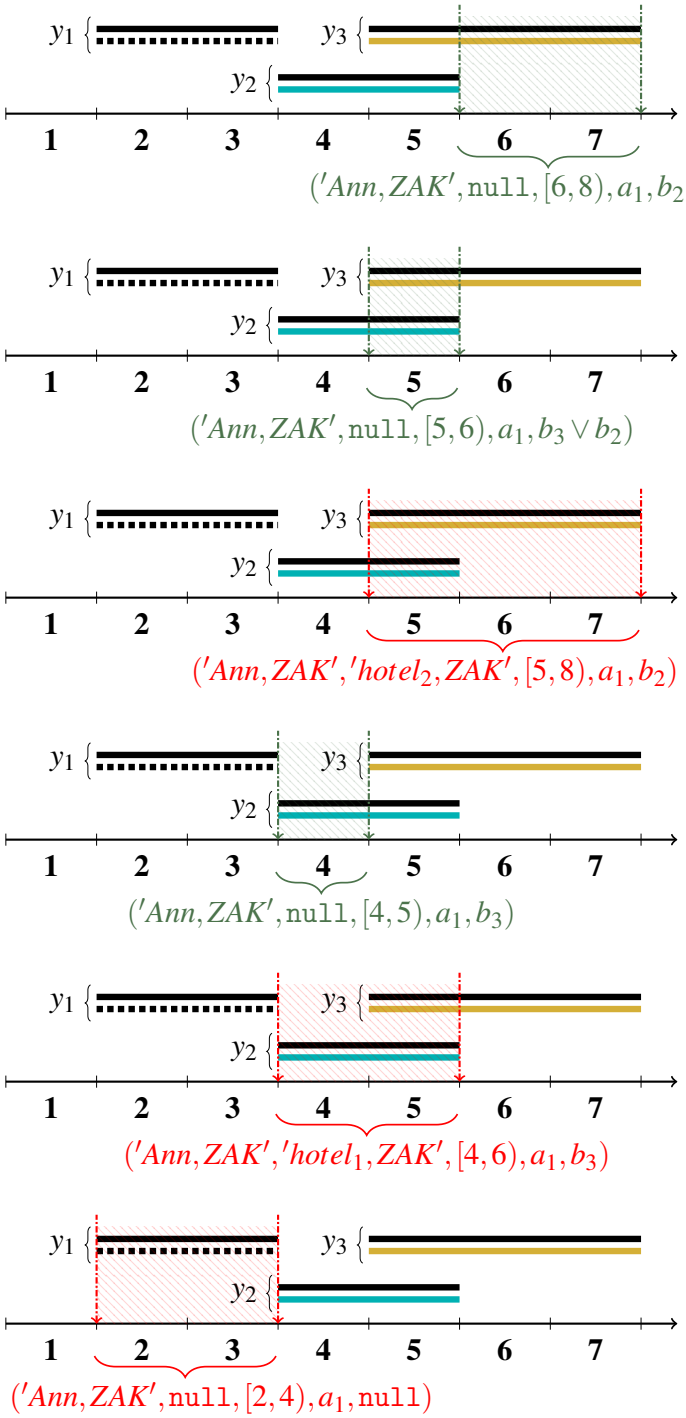


Figure 3.10: Execution of LAWA_{N} on the result of LAWA_{U}

the next negating window is set to *windTe*. neg is set to $false$ so that the window wind is copied to the output.

A special case occurs when the starting point of the upcoming window is equal to the starting point of the output window (Line 25. This means that there exists a valid tuple in the reference relation **s** that needs to be considered for the output window and thus its finalization is postponed. The upcoming window, either overlapping or unmatched, has to be first copied to the output so we set neg back to false.

If there are more overlapping windows in PQ that end before the upcoming window wind starts, i.e., wind.Ts < tForTopOfPQ(), regardless of whether wind belongs in the same or a different group, the ending point of the new negating window is equal to the ending point of the pair on top of the priority queue. The starting point of the next negating window is set to windTe indicating that the sweeping until this time point has been completed. As a result all the nodes in the priority queue corresponding to windows whose ending point is equal to windTe have already been considered and need to be removed.

Example 16. In Fig. 3.10, we illustrate all six calls of LAWA_{N} on the result **Y** (Fig.3.9) of LAWA_{UO} of the relations **a** and **b** of Fig.3.1a. Red color is used for the unmatched and overlapping windows copied to the output whereas green is used for the negating windows. In the first two calls of LAWA_{N}, illustrated at the bottom, two unmatched-interval windows of the group related to F='Ann, ZAK' are copied to the output. The overlapping window y_3 is fetched and it is the first overlapping window after a series of unmatched ones. Thus prevWindTe is set to $y_3.Ts = 4$, F = 'Ann, ZAK' and $\lambda_r = r_2$. After $out = y_3$, neg is set to $true$ and the sweeping for negating tuples starts from prevWindTe $= 4$. Window y_3 is followed by another overlapping window (y_4) that starts before the ending point of y_3, recorded in the top node of the priority queue. Consequently, windTe $= y_4.Ts = 5$ and the negating window ('Ann, ZAK', null, [4, 5), a_1, b_3) is produced and neg is set false. Window y_4 is then copied to the output and since there are no more windows to be processed the ending points for the remaining negating windows are derived solely from the list PQ.

3.7.4 TP Join Algorithms

In this subsection we introduce the algorithm *NegationJoins*(**r**, **s**, θ, op) that computes the result of the TP operation with negation op based on the input TP relations **r** and **s** and the predicate θ. In contrast to previous works in either temporal or probabilistic databases, this algorithms

involves no tuple replication. It also allows for a pipelined calculation of the results for all TP joins with negation and thus enables its smooth integration in the kernel of a DBMS.

Algorithm 7: *NegationJoins*(**r**, **s**, θ, op)

```
1  w_init = leftJoin(r, s, θ ∧ θ_overlap);
2  sort(w_init{F_L, O_s}) ;
3  status_u = (−1, null, null, null, fetchWindow(w_init));
4  while status_u ≠ null do
5  |   (w, status_u) = LAWA_u(status_u);
6  |   w_uo = w_uo ∪ {w};
7  status_n = (−1, null, null, null, false, fetchWindow(w_uo));
8  while status_n ≠ null do
9  |   (w, status_n) = LAWA_n(status_n);
10 |   if w.λ_s = null ∧ w.F_s = null then
11 |   |   o = o ∪ {(w.F_r, w.F_s, w.λ_r, [w.winTs, w.winTe))};
12 |   else if w.λ_s ≠ null ∧ w.F_s = null then
13 |   |   λ = andNot(w.λ_r, w.λ_s);
14 |   |   o = o ∪ {(w.F_r, w.F_s, λ, [w.winTs, w.winTe))};
15 |   else if op ≠ ▷ then
16 |   |   λ = and(w.λ_r, w.λ_s);
17 |   |   o = o ∪ {(w.F_r, w.F_s, λ, [w.winTs, w.winTe))};
18 if op = ⟗ then o = o ∪ NegatingJoins(s, r, θ, ▷) ;
19 return o;
```

Initially, the set $\mathbf{w_{init}}$ is produced using the conventional left outer join described in Section 3.7.1 that includes all overlapping windows of **r** and **s** as well as a subset of the unmatched windows. The windows of $\mathbf{w_{init}}$ are sorted based on the facts F_r and the start points Ts (Line 2) that correspond to tuples of the positive relation from which they have been produced. As long as the terminating condition (Line 8) is satisfied, LAWA$_u$ passes through all start and end points of the windows in $\mathbf{w_{init}}$ in a smaller-to-larger fashion and expands the set with the unmatched windows (Line 6) that hadn't been created yet. Similarly, LAWA$_n$ sweeps the windows of the set $\mathbf{w_{uo}}$ and extends it with the negating windows of **r** and **s**.

Each window w that LAWA$_n$ produces is not further swept and so it can be transformed to an output tuple for the result of the TP join. A lineage-based filter is directly applied to determine if the window returned is unmatched ($\mathtt{w}.\lambda_s = \mathtt{null} \wedge \mathtt{w}.F_s = \mathtt{null}$), negating ($\mathtt{w}.\lambda_s \neq \mathtt{null} \wedge \mathtt{w}.F_s = \mathtt{null}$) or overlapping. If the operation performed is a TP anti-join ($\triangleright^{TP}$), then the overlapping windows are filtered out and are not included in the final result. If it is a full outer-join, the unmatched and negating windows of **s** using **r** as a reference need to be included and thus

the *NegationJoins* algorithm needs to be called again with reversed arguments, same predicate and anti-join as the operation to be performed so that the overlapping windows are not copied again to the output. Finally, every window is finalized into an output tuple using the lineage-concatenating function that corresponds to the TP join computed.

3.8 Evaluation

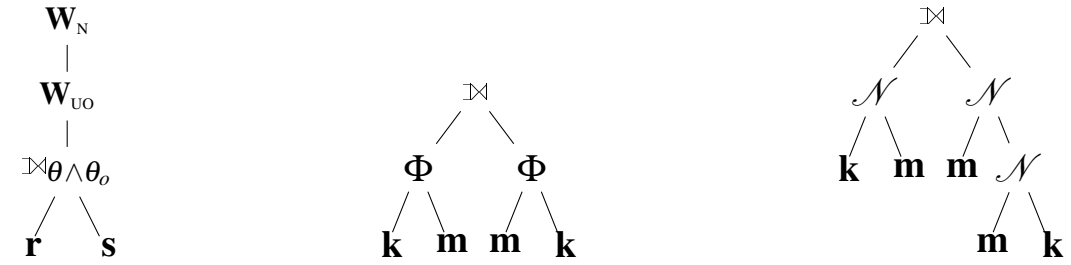
In this section, we evaluate our algorithms using two real-world datasets which vary on (i) the number of facts in the input relations and (ii) the percentage of tuples whose intervals overlap. We compare our approach for TP joins with negation (NJ) to Temporal Alignment (TA), i.e., the only related approach that can be used for the computation of TP outer joins and TP anti join. The experiments show that our approach outperforms TA and it is the only scalable solution for TP joins with negation on input relations of more than 200K tuples. *NJ* is also robust with predictable performance with respect to the aforementioned characteristics of the datasets.

3.8.1 Experimental Setup

All of the following experiments were deployed on a 2xIntel(R) Xeon(R) CPU E5-24400 @2.40GHz machine with 64GB main memory, running CentOS 6.7. Our algorithms have been implemented in the kernel of PostgreSQL in C, and all experiments were performed in main-memory. No indexes were used. In all PostgreSQL implementations, the maximum memory for sorting as well as for shared buffers were set to 10GB.

We have implemented *NJ* in PostgreSQL 9.4.3 by modifying the parser, executor and optimizer. The only approach our implementation can be compared against is **Temporal Alignment (TA) [DBGJ16b]**. **Temporal Alignment** is an approach developed for the computation of temporal operations using sequenced semantics and is implemented in the kernel of PostgreSQL as well. It consists of a set of reduction rules based on *Normalize* ($\mathscr{N}$) and *Align* (Φ), two operators responsible for the interval adjustment of the input relations. Due to the existence of probabilities, the results of TP joins with negation differ and thus, for our experiments, we introduced reduction rules that are consistent with the TP semantics while properly exploiting $\mathscr{N}$ and Φ. For a fair comparison, we migrated the authors' implementation to PostgreSQL 9.4.3.

In Fig. 3.11, we illustrate the query plans used by NJ and TA for the computation of windows. In Fig. 3.11a, the nodes $\mathtt{w_{init}}$, $\mathtt{w_{uo}}$ in the tree correspond to sets of windows as described in Algorithm **??**. The node $\mathtt{w_N}$ corresponds to the set of negating windows produced by the calls of LAWA$_N$. In Fig. 3.11b and 3.11c, we illustrate the two query subtrees in TA for the computation of all output tuples. The operators $\mathscr{N}$ and Φ in TA replicate the tuples of the left relation and assign new intervals based on the right relation. Since the facts and lineages of the input tuples still need to be combined, additional joins are performed. $\Phi(\mathbf{k},\mathbf{m})$ is associated with overlapping windows (Fig. 3.11b) since the subintervals it produces correspond to the overlap of a tuple in $\mathbf{k}$ with a tuple in $\mathbf{m}$. $\mathscr{N}(\mathbf{k},\mathbf{m})$ is appropriate for negating windows since it includes intervals that correspond to the overlap of a tuple in $\mathbf{k}$ with a group of tuples in $\mathbf{m}$. Both $\Phi(\mathbf{k},\mathbf{m})$ and $\mathscr{N}(\mathbf{k},\mathbf{m})$ include intervals where a tuple k in $\mathbf{k}$ matches no tuple in $\mathbf{m}$, leading to the unmatched windows being computed twice. In Fig. 3.11c, the tuples of the right relation $\mathbf{m}$ are adjusted both using relation $\mathbf{k}$ and itself because, over an interval, we compute the tuples of $\mathbf{m}$ that are valid and are combined with a tuple of $\mathbf{k}$. Given that $\mathscr{N}$ only uses one input relation as reference, we need to further adjust $\mathbf{m}$ based on the result of $\mathscr{N}(\mathbf{k},\mathbf{m})$.



(a) NJ: all (b) $TA : \mathbf{W}_{\mathrm{O}} \cup \mathbf{W}_{\mathrm{U}}$ (c) $TA : \mathbf{W}_{\mathrm{N}} \cup \mathbf{W}_{\mathrm{U}}$

Figure 3.11: Query Trees

The $\mathbb{N}_{\theta\wedge\theta_o}$, $\mathscr{N}$ and Φ nodes are all based on a conventional left-outer join with a condition for the interval overlap of the matching tuples. PostgreSQL's optimizer determines whether such a join is executed as a nested loop, a merge join or a hash join depending on the θ codition of the TP join to be computed. $\mathbb{N}_{\theta\wedge\theta_o}$ is computed using a nested loop only when the θ condition used has low selectivity, i.e., when a high percentage of pairs of input tuples satisfy the condition. On the contrary, this varies for $\mathscr{N}$ and Φ, based on whether a TP join or a set of windows is computed.

3.8.2 Real-World Datasets

The Webkit dataset[1] [DBG14, PHD16, CB17] records the history of 484K files of the SVN repository of the Webkit project over a period of 11 years at a granularity of milliseconds. Each tuple has schema (*File_Path, [T_s, T_e)*) and the valid times indicate the periods when a file remained unchanged. The Meteo Swiss dataset[2] includes temperature predictions that have been extracted from the website of the Swiss Federal Office of Meterology and Climatology. Each tuple has schema (*Station_ID, Value_ID, Value, [T_s, T_e)*). The measurements were taken at 80 different meteorological stations (Station_ID) in Switzerland from 2005 to 2015 and involve four different metrics (Value_ID), including temperature and precipitation. Measurements are 10 minutes apart and – in order to produce intervals – we merged time points whose measurements differ by less than 0.1.

Table 3.4: Real-World Dataset Properties

	Meteo	**Webkit**
Cardinality	10.2M	1.5M
Time Range	347M	7M
Min. Duration	600	0.02
Max. Duration	19.3M	6M
Avg. Duration	152M	1.7M
Num. of Facts	80	484K
Distinct Points	545K	144K
Max Num. of Tuples (per time point)	140	369K
Avg Num. of Tuples (per time point)	37	21

The main properties of these datasets are summarized in Table 3.4. For both datasets we produced a second relation by shifting the intervals of the original dataset, without modifying the lengths of the intervals. The start/end points of the new relation were chosen according to the distribution of the original ones.

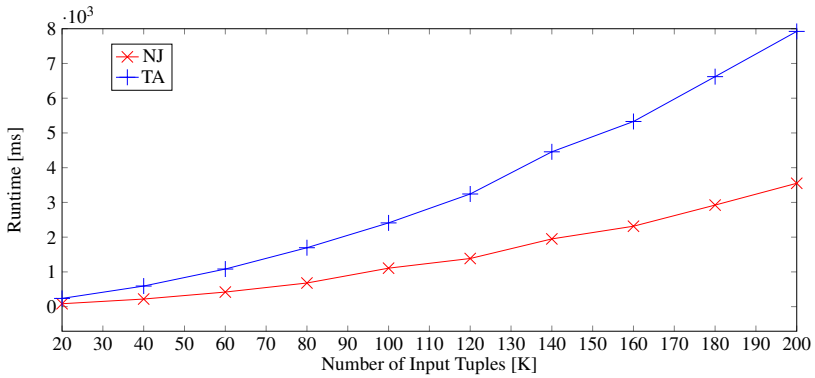
3.8.3 Runtime

In Fig. 3.12, 3.13, 3.14 we illustrate the runtime for the overlapping and unmatched windows, negating windows, and for a TP left outer join, respectively, over subsets of the Webkit and Meteo dataset. The subsets range from 20K to 200K tuples. For Webkit dataset, as a θ condition we apply equality of the *File_Path*, i.e., we combine tuples referring to the same file. For Meteo

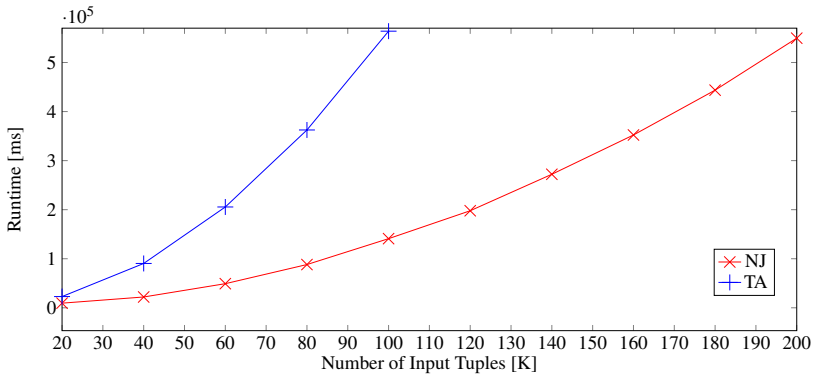
[1] The WebKit Open Source Project: http://www.webkit.org (2012)
[2] Federal Office of Meteorology and Climatology: http://www.meteoswiss.ch (2016)

dataset, we apply equality on *Value_IDs* and inequality on *Station_IDs*, i.e. we combine tuples with measurements on the same metric but taken in different stations.

Fig. 3.12 shows the runtime of NJ and TA for the set $\mathtt{w}_{\text{UO}}$ (Algorithm **??**), including the unmatched and overlapping windows. Both approaches follow a similar trend and the reason is that the most computationally demanding part of both is a conventional left join, used to identify the pairs of tuples that overlap. As shown in Fig. 3.11, NJ only executes this join once whereas TA executes it twice. As a result, NJ is two to four times faster.



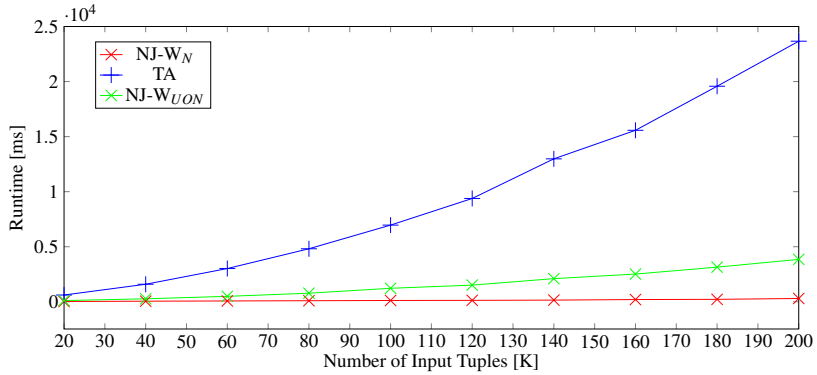
(a) Webkit



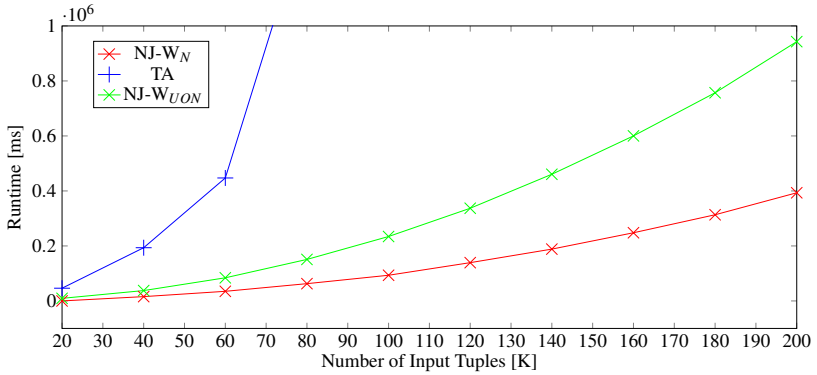
(b) Meteo

Figure 3.12: W_{UO}: Overlapping and Unmatched Windows

In Fig. 3.13, we have illustrated the runtime for the computation of negating windows. In NJ, negating windows are computed by applying $LAWA_{\text{N}}$ on the set $\mathtt{w}_{\text{UO}}$. Thus, we have illustrated their computation time both including (W_{UON}) and excluding (W_N) the runtime for $\mathtt{w}_{\text{UO}}$. In the case of W_{UON}, NJ computes the negating windows four to ten times faster than TA whereas, in the case of W_N, it computes them twelve to twenty times faster.



(a) Webkit

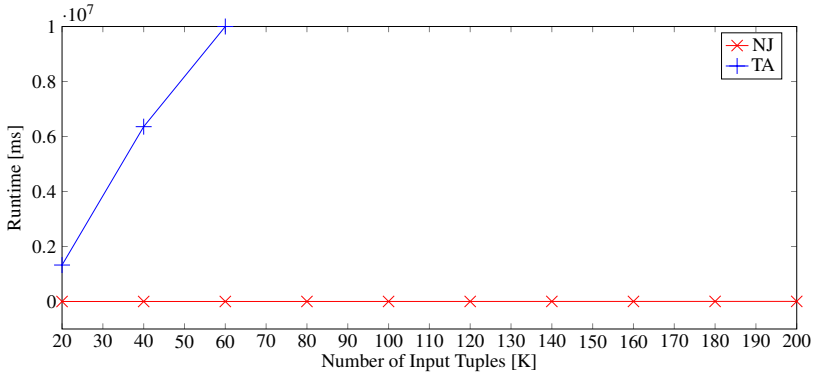


(b) Meteo

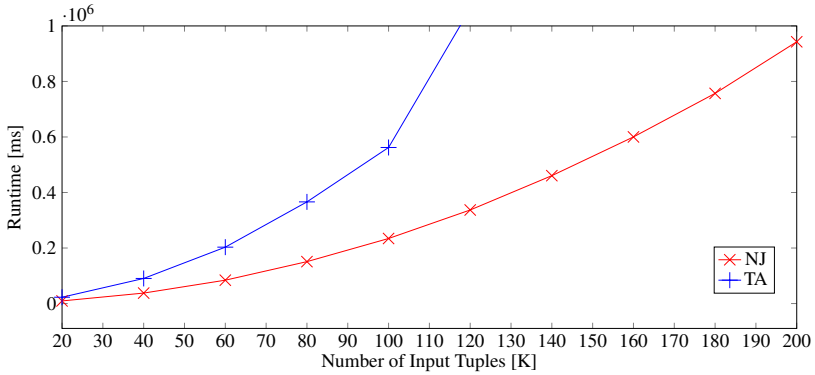
Figure 3.13: Negating Windows

Finally, the runtimes of both NJ and TA for a TP left-outer join are illustrated in Fig. 3.14. To compute the join with TA, a duplicate-eliminating is applied on the query trees in Fig. 3.11b and Fig. 3.11c to combined the partial results and remove the redundant unmatched windows. Its runtime for the TP left-outer join is much higher than the sum of the runtimes of the windows as presented in Fig. 3.12 and Fig. 3.13. The reason for that is that when the union of the query trees in Fig. 3.11b and 3.11c is performed, the θ condition of the TP join is ignored for the right subtree of Fig. 3.13. The optimizer opts for a nested loop for its computation and this takes a huge toll on TA's runtime making NJ two orders of magnitude faster.

Meteo dataset contains a number of distinct values much smaller than its size, an analogy maintained in the subsets due to the use of the uniform distribution in their creation. As a result, the condition is not very selective and the runtime of both NJ and TA is higher than it was in the case of the webkit dataset. In all cases, the runtime of NJ outperforms TA by four to ten times.



(a) Webkit



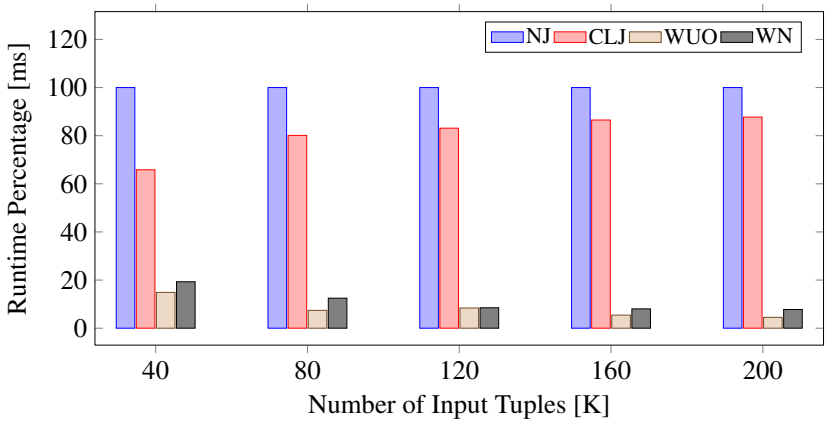
(b) Meteo

Figure 3.14: TP Left Outer-Join

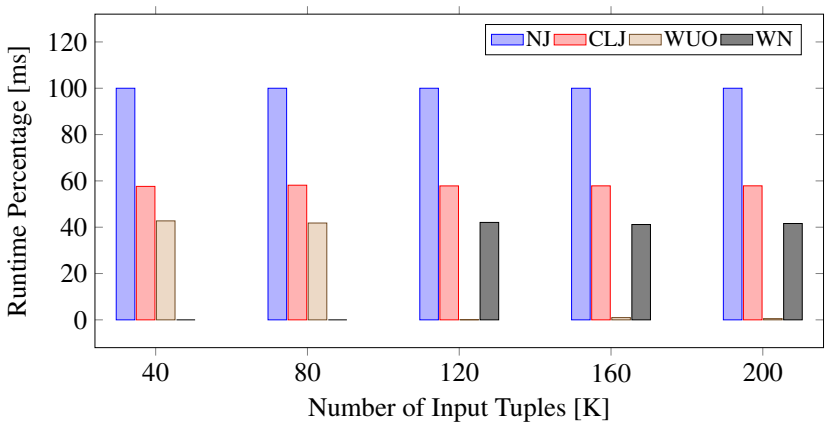
3.8.4 Runtime Breakdown and Scalability

The query tree of the NJ approach (cf. Fig. 3.11a) consists of the nodes $\rtimes_{\theta \wedge \theta_o}$, $\mathscr{W}_{uo}$ and $\mathscr{W}_n$ nodes. The way that the node $\rtimes_{\theta \wedge \theta_o}$ is computed is completely determined by PostgreSQL's optimizer, given the condition applied on the non-temporal attributes. The most demanding part of the node $\mathscr{W}_n$ is handling the tuples valid over the interval of the window. In Fig. 3.15, we breakdown the runtime of a TP left outer join on the percentage occupied by each node of the query tree for Webkit and Meteo dataset, respectively. As shown in the graphs, the conventional left-outer join (CLJ) occupies most of the runtime of the TP left outer join (NJ) which is more than 50% for Webkit dataset. The calls to LAWA_U and LAWA_N, for the computation of the nodes $\mathscr{W}_{uo}$ and $\mathscr{W}_n$ respectively, correspond to a small percentage of the runtime in Webkit dataset. However, they tend to be more time-consuming for Meteo dataset. This behaviour lies in the dataset characteristics and in the query performed. In meteo, the θ condition used requests for the tuples combined to have the same metric but to refer to different stations. Measurements over

all stations take place at similar times and, for multiple output intervals, all valid tuples might contribute in the output, making the computations much more demanding.



(a) Webkit Dataset.



(b) Meteo Dataset.

Figure 3.15: Runtime Breakdown and Scalability. CLJ corresponds to $\mathbb{N}_{\theta \wedge \theta_o}$ and NJ to $\mathbb{N}^{\mathrm{Tp}}_{\theta \wedge \theta_o}$.

NJ is the only scalable approach integrated in PostgreSQL that can be used for the computation of all TP joins including negation. In Fig. 3.16, we depict the performance of NJ for the computation of a TP left outer join for larger subsets of the webkit and meteo datasets. *TA* is not taken into consideration, since its runtimes were already one to four orders of magnitude higher than NJ's when applied on the smaller datasets. The dataset sizes vary from 100K to 1M tuples. NJ's implementation is based on a conventional left outer join and its performance is influenced by the condition on the non-temporal attributes, since the optimizer opts for a different type of join. The selectivity of the condition applied in the webkit dataset is higher, allowing for the computation of the left outer join using a merge join. On the contrary, in the case of meteo dataset, a nested loop has to be computed. As a result, NJ scales more efficiently when applied

on the webkit dataset, with its runtime being two minutes on average and always less than five minutes for datasets less than 2M.

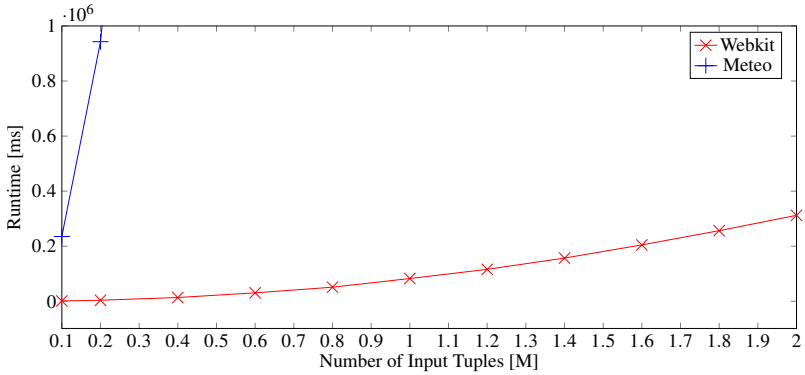


Figure 3.16: Scalability

3.9 Conclusions

In this work, we proposed an approach for the computation of temporal-probabilistic joins with negation, operations that cannot currently be performed by any existing TP approach. We introduced the generalized lineage-aware temporal windows, to bind lineages and intervals and comply with requirements of TP joins. These windows we split to three sets and using these sets we express the result of each TP join with negation. We implemented algorithms for the pipelined computation of all sets of generalized lineage-aware temporal window and we integrated our approach in the kernel of PostgreSQL. A thorough experimental evaluation reveals that our implementation is seamlessly integrated in the DBMS and outperforms existing approaches.

CHAPTER 4

TemProRA: Top-k Temporal-Probabilistic Results Analysis

Abstract

The study of time and probability, as two combined dimensions in database systems, has focused on the correct and efficient computation of the probabilities and time intervals. However, there is a lack of analytical information that allows users to understand and tune the probability of time-varying result tuples.

In this demonstration, we present TemProRA, a system that focuses on the analysis of the top-k temporal probabilistic results of a query. We propose the Temporal Probabilistic Lineage Tree (TPLT), the Temporal Probabilistic Bubble Chart (TPBC) and the Temporal Probabilistic Column Chart (TPCC): for each output tuple these three tools are created to provide the user with the most important information to systematically modify the time-varying probability of result tuples. The effectiveness and usefulness of TemProRA are demonstrated through queries performed on a dataset created based on data from Migros, the leading Swiss supermarket branch.

4.1 Introduction

Time and probability have been studied as separate dimensions in database systems, with the work of Dylla et al. [DMT13] being a notable exception. Various temporal [DBG12, DBG12, AGC+13, KVF+13] and probabilistic [STW08, AKO07, BDM+05, OHK09] extensions of DBMSs have been implemented. The focus has been the correct and efficient computation of the probabilities and/or time intervals. The interpretation and use of the result tuples has received scant attention. In most cases additional information is needed to properly interpret result tuples and be able to systematically modify the probabilities of the base tuples.

Table 4.1: The Top-3 Result Relation for Q_1

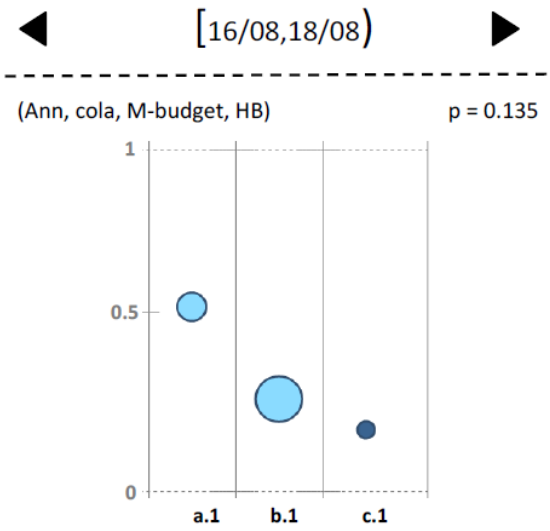
Q_1 (wantsToBuyIn_IsNotAvailable)

	N	*P*	*B*	*L*	*T*	*p*
q_{11}	Ann	cola	M-budget	HB	[14/08,16/08)	0.15
q_{12}	Ann	salad	M-budget	HB	[15/08,24/08)	0.20
q_{13}	Ann	cola	M-budget	HB	[16/08,18/08)	0.135

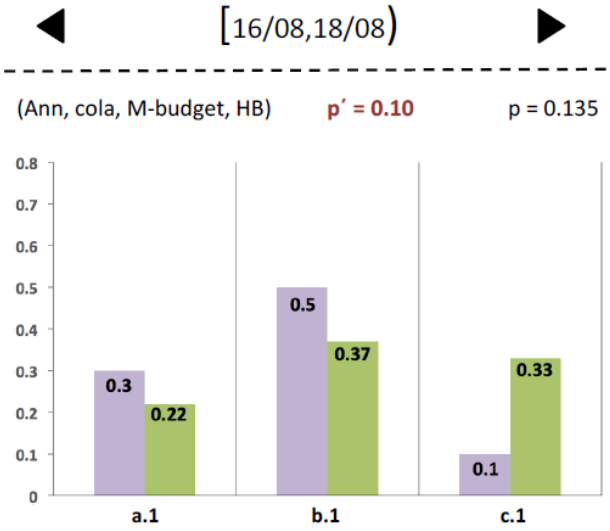
Example 17. *Assume that Migros intends to calculate and tune the probability that a customer wants to buy a product that is not available. The three result tuples with the highest probabilities for the query Q_1 = "At each time point in August, what is the probability that Ann wants to buy a product that is not in stock in a Migros location where she shops?" are illustrated in Table 4.1. Tuple q_{13} indicates that, at each time point in the time interval [16/08,18/08), with probability 0.135, Ann wants to buy cola from the brand M-budget at location HB, and cola is not in stock. If Migros wants to decrease this probability, the information in q_{13} is clearly insufficient. It is not possible to precisely estimate the probability of q_{13} if the probability that "Ann wants to buy cola from brand M-budget" is increased. Similarly, it is unclear whether a larger decrease is achieved by modifying the probability that "Ann shops in HB" or by modifying the probability that "Ann wants to buy cola from brand M-budget".*

TemProRA resolves the issues raised in the above example. It focuses on the analysis and tuning of the k result tuples of a temporal-probabilistic query with either the lowest or the highest probabilities. More specifically, we propose the following key tools to analyze each result tuple s (Fig. 4.1):

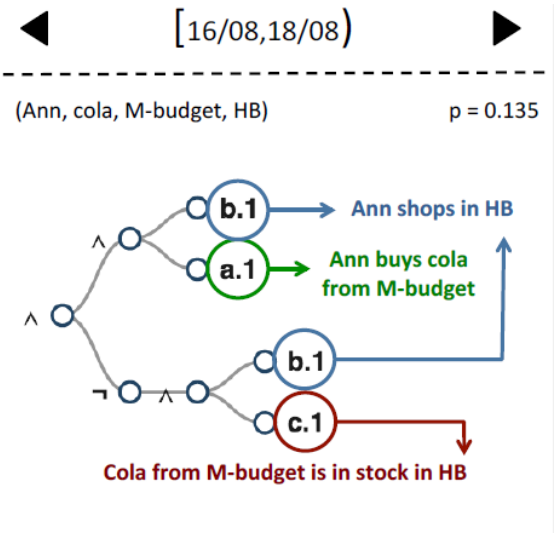
- **The Temporal Probabilistic Bubble Chart (TPBC):** The TPBC illustrates the impact that a change in the probability of base tuples has on the probability of s. It focuses on the tuples with the highest impact and encodes whether it is positive or negative as well as its magnitude.



(a) TPBC



(b) TPCC



(c) TPLT

Figure 4.1: The Features of TemProRA on Tuple q_{13}.

- **The Temporal Probabilistic Column Chart (TPCC):** Assuming that the probability of s is to be modified from $s.p$ to $s.p'$, the TPCC reports how this goal is achieved via the modification of the probability of its most influential tuples (the ones with the highest impacts).

- **The Temporal Probabilistic Lineage Tree (TPLT):** The TPLT visualizes the lineage of s, allowing the identification of all tuples influencing its probability. For each point in time, the TPLT shows how and from which base tuples s has been derived.

4.2 Related Work

In this section, we discuss related works that propose or illustrate additional information to interpret result tuples. The first effort towards this direction has been made in the context of the Trio Project via Trio Explorer [MTea07, Wid09]. Trio Explorer is an API paired with the Trio system that presents the lineage of the alternatives of a tuple to the user. Trio does not quantitatively connect the probability of an input tuple t with the probability of an output tuple s.

The impact of each base tuple on the probability of an output tuple has been used for the computation of a proper probability threshold in top-k processing in probabilistic databases [DMT13, RDS07, KLD11]. The goal in these works has been the computation of the result relation and the impact of a tuple was not used further.

4.3 Preliminaries

A *temporal probabilistic schema* is denoted by $R^{TP}(\mathbf{A}, T, p)$ where $\mathbf{A}$ is a set of conventional attributes, $T = [t_s, t_e)$ is a temporal attribute in the form of a time interval, with the convention that t_s is the inclusive start and t_e the exclusive end point, and p is a probabilistic attribute over $(0, 1]$. Given a *tuple* r over $R^{TP}(\mathbf{A}, T, p)$, an event e refers to its set of conventional attributes, also noted as $r.\mathbf{A}$. Semantically, r indicates that at each time point included in the interval T, the event e is true with probability p. We assume *a tuple independent database*, i.e., the probability

assigned to a base tuple (a tuple stored in the database) does not depend on any of the other base tuples.

Table 4.2: The Supermarket Application Scenario

A (buys)

k	N	P	B	T	p
a_1	Ann	cola	M-budget	[13/08,18/08)	0.3
a_2	Liz	muesli	Farmer	[20/08,27/08)	0.2
a_3	Ann	salad	M-budget	[15/08,24/08)	0.4
a_4	Ann	chips	M-budget	[12/08,19/08)	0.1

B (shopsIn)

	N	L	T	p
b_1	Ann	HB	[14/08,26/08)	0.5
b_2	Liz	OE	[21/08,23/08)	0.6

C (inStockIn)

	P	B	L	T	p
c_1	cola	M-budget	HB	[16/08,25/08)	0.1
c_2	muesli	Farmer	OE	[27/08,29/08)	0.7
c_3	chips	M-budget	HB	[04/08,23/08)	0.3

Example 18. *Consider Migros, the leading supermarket branch in Switzerland. In Table 4.2, relation **A** contains the products that customers buy during August, relation **B** contains the locations where customers shop, and relation **C** contains the products that are in stock in each store. Abbreviations HB and OE refer to the stores in Hauptbahnhof (Central Station) and in Oerlikon, respectively. M-budget and Farmer are two of the brands with products sold in Migros. Tuple* a_1 *from relation* **A** *(Table 4.2) indicates that, at each day from August 13 until August 18, the event "Ann shops cola from M-budget" is true with probability 0.3.*

Result computation in TPDBs is conducted using data lineage. Lineage is used to properly determine the time points included in the time interval attached to a tuple [DBG12], but also to correctly compute the probability ([STW08, OHK10]). It is a *Boolean formula* consisting of tuple identifiers, corresponding to random variables, connected with logical symbols $(\wedge, \vee, \neg)$. The lineage of a base tuple is equal to its identifier, e.g., $\lambda(a_1) = a_1$. The identifiers and logical symbols in the lineage of an output tuple are dictated by the operator applied and the base tuples participating in the operation.

Example 19. *Tuple q_{13} (Fig. 4.1) is included in the result of query Q_1 evaluated using the expression $(A \bowtie^{TP} B) -^{TP} (B \bowtie^{TP} C)$, where A,B,C are the relations in Table 4.2. The lineage of q_{13} (Fig. 4.1) is $\lambda(q_{13}) = (a_1 \wedge b_1) \wedge \neg(b_1 \wedge c_1)$ (Fig.4.1). The base tuples involved are a_1, b_1 and c_1. Thus, cola from M-budget not being available in HB, when Ann wants to buy it, depends on Ann buying cola from M-budget(a_1), on her shopping in HB (b_1), and on cola being in stock in HB (c_1).*

4.4 Temprora Architecture and Features

In the backend of TemProRA lies a temporal-probabilistic extension of PostgreSQL, responsible for the query evaluation. It correctly deals with the conventional attributes, the time interval and the probability value through the combination of timestamp alignment [DBG12], data lineage [DMT13, STW08], and rules that systematically reduce each temporal-probabilistic operator to its conventional counterpart. The front-end visualisation of TemProRA is a Web Application created using the "D3.js" library. The user inserts a query in temporal probabilistic (TP) algebra or SQL and s/he is presented with the result evaluated in the backend. When a result tuple s is selected, the three main features of TemProRA are produced:

The Temporal Probabilistic Bubble Chart (TPBC) of an output tuple s illustrates the k base tuples in $\lambda(s)$ with the highest impacts on $s.p$. The x-axis of the chart is divided in columns, each corresponding to a tuple t in $\lambda(s)$. The y-axis represents the value of the probability impact (u_t), as described in Section 4.5. The color represents the sign of u_t: dark blue if $u_t < 0$ and light blue if $u_t > 0$. The diameter of a bubble is proportional to $t.p$ so that the user can decide if $t.p$ can/should be modified or not.

The Temporal Probabilistic Column Chart (TPCC) supports the modification of the probability of an output tuple s from $s.p$ to $s.p'$, where $s.p'$ is a value determined by the user. The x-axis of the TPCC is divided into multiple columns, each corresponding to a tuple in the TPBC. The y-axis displays probability values. For each tuple t included in the TPBC of s, the TPCC uses a vertical purple bar with height equivalent to its current probability $t.p$ and a vertical green bar with height equivalent to the value $t.p'$ that would guarantee the modification of $s.p$ to $s.p'$.

The Temporal Probabilistic Lineage Tree (TPLT) of a result tuple is a representation of its lineage expression. Its purpose is to provide insights for experienced users. It illustrates the complete lineage expression of a result tuple s in a tree form and, thus, the dependencies involved

in the probability computation process as well as all the input tuples influencing $s.p$ can be identified. The leaves of the tree correspond to base tuples and are labeled with tuple identifiers, whereas intermediate nodes correspond to logical operators.

As an example, the TPBC, TPCC and TPLT of tuple q_{13} (Fig. 4.1) are illustrated in Fig. 4.1. At the top of all three tools, the event described by q_{13} and its time interval are noted. With the help of the arrows on both sides of the interval, the user is given the opportunity to observe the evolution of the probability of this event over time. This form of "time-travel" reveals that the input tuples affecting the probability of an event might vary over different time points.

4.5 The Time-Varying Probability Impact

Within the interval $s.T$, the exact probability $s.p$ of an output tuple s is computed via the arithmetic valuation of its lineage expression $\lambda(s)$ [DMT13, STW08]. It is defined as a sum of products where each product consists of terms of the form $t_k.p$ or $1-t_k.p$ for each base tuple t_k appearing in $\lambda(s)$. Regrouping the sum based on this criterion, the probability $s.p$ can be expressed as a linear function of $t_k.p$ [DMT13].

Given that $s.p = u_k \cdot t_k.p + v_k$, the probability impact u_k of t_k quantifies how much the increase/decrease of $t_k.p$ affects $s.p$. Given that $t_k.p, s.p \in (0,1]$ and the linear relation between them, the values that $s.p$ can acquire via a modification of $t_k.p$ are restricted within the interval $(v_t,\ u_t + v_t]$.

The coefficients u_t and v_t are the main concept in the creation of the TPBC and TPCC. They are determined based on two pairs of values ($t_k.p$,$s.p$) in the function. In our computations, we use the pairs (t_k^{init},$s.p^{init}$) and (1,$s.p^1$), where $s.p^{init}$ is the probability of tuple s based on the initial value of $t_k.p$ and $s.p^1$ is the probability of s assuming that $t_k.p = 1$.

Proposition 3. *Assume output tuple s and base tuples t_i, t_j that appear in $\lambda(s)$ and for which $s.p = s.p_i = u_i \cdot t_i.p + v_i$ and $s.p = s.p_j = u_j \cdot t_j.p + v_j$, with $u_i > u_j$ and u_i, u_j having the same sign. A modification in $t_i.p$ such that $t_i.p' = t_i.p + c$, yields a greater modification in $s.p$ than the same modification of $t_j.p$.*

The pair $(t_i.p, s.p)$ satisfies $s.p_i = u_i \cdot t_i.p + v_i$ whereas $(t_j.p, s.p)$ satisfies $s.p_j = u_j \cdot t_j.p + v_j$. Initially, $s.p = s.p_i = s.p_j$ so $u_i \cdot t_i.p + v_i = u_j \cdot t_j.p + v_j$ (1). Assume $t_i.p, t_j.p$ are increased by $c > 0$ so that $t_i.p' = t_i.p + c$ and $t_j.p' = t_j.p + c$ yield $s_i.p'$ and $s_j.p'$. Based on equation (1) and on the linear relation between $t_i.p'$ and $s_i.p'$ as well as between $t_j.p'$ and $s_j.p'$: $s.p'_i - s.p'_j =$

$(u_i - u_j) \cdot c$. Since $c > 0$, $u_i > u_j$ and u_i, u_j have the same sign, we conclude $s.p'_i > s.p'_j$. Similarly, the result of the proposition is explained for $c < 0$.

When regrouping the sum of products used to define the probability of a result tuple s with respect to the probability of the base tuple t_k, the coefficients of t_k are influenced by the probabilities of the other based tuples in $\lambda(s)$ other than $t_k.p$. Accordingly, a modification in $t_k.p$ is expected to alter the values of the probability impacts of the base tuples in $\lambda(s)$.

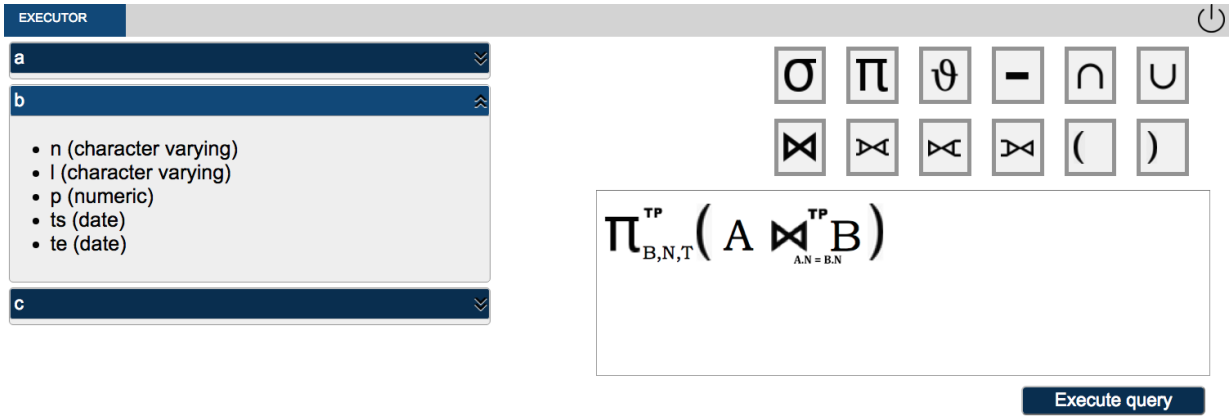


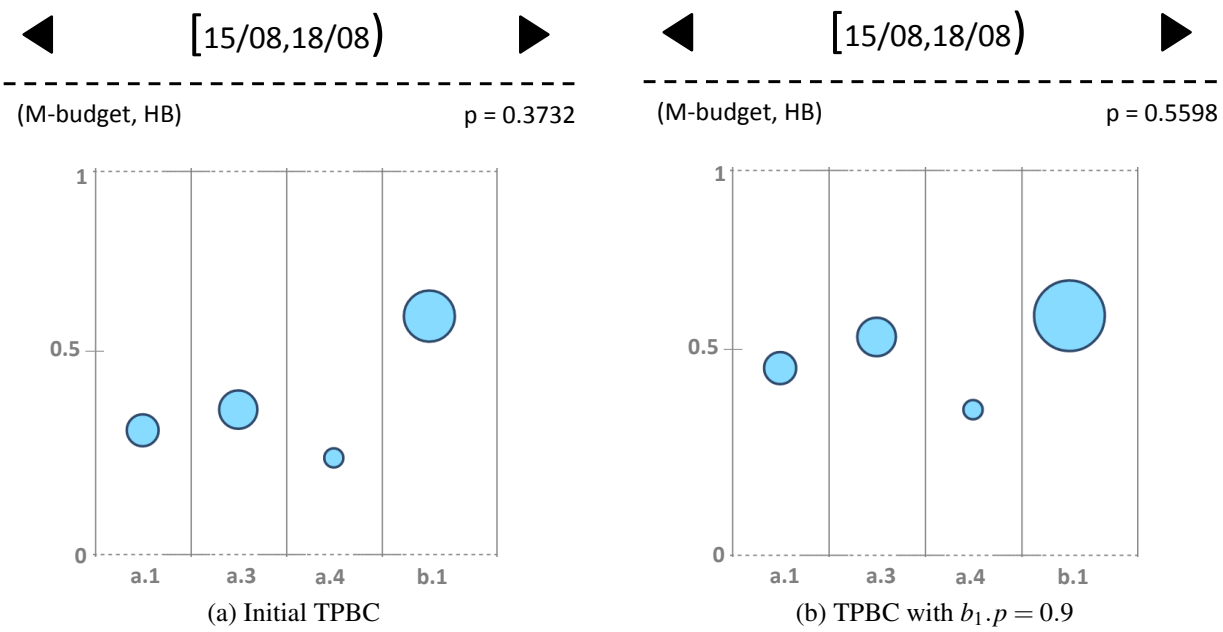
Figure 4.2: Result Demonstrator

4.6 Demonstration Scenario

During the demonstration, the attendees are encouraged to use a set of the predefined queries over a supermarket dataset created based on data from the Swiss supermarket branch Migros. Each query corresponds to a combination of operators (e.g., projections over joins) where determining the base tuples that influence a result tuple is non-trivial. All queries are illustrated both in temporal-probabilistic algebra and SQL. They are evaluated in the backend and the top k result tuples ($k \in [0, 50]$) are presented and further analysed using the features of TemProRA.

As a demonstration scenario, we use relations A, B, C (Table 4.2) to evaluate query Q_2 = *"At each time point, what is the probability that a customer buys at least one product from brand M-budget in a Migros location where they shop?"*, useful for a possible promotion of the brand. Fig. 4.2 captures the insertion of the algebra expression corresponding to query Q_2 together with its result.

After the result has been produced, the user clicks on one of the tuples in the result, whose probability they wish to modify, either in an increasing or decreasing manner. The click of the tuple triggers the creation of the corresponding TPBC, TPCC and TPLT. Assume the chosen tuple is q = (*M-budget, [15/08,18/08), 0.3110)*, stating that during *[15/08,18/08)*, with probability *0.3110*, at least one product from brand *M-budget* is bought in the Migros store in *HB*.

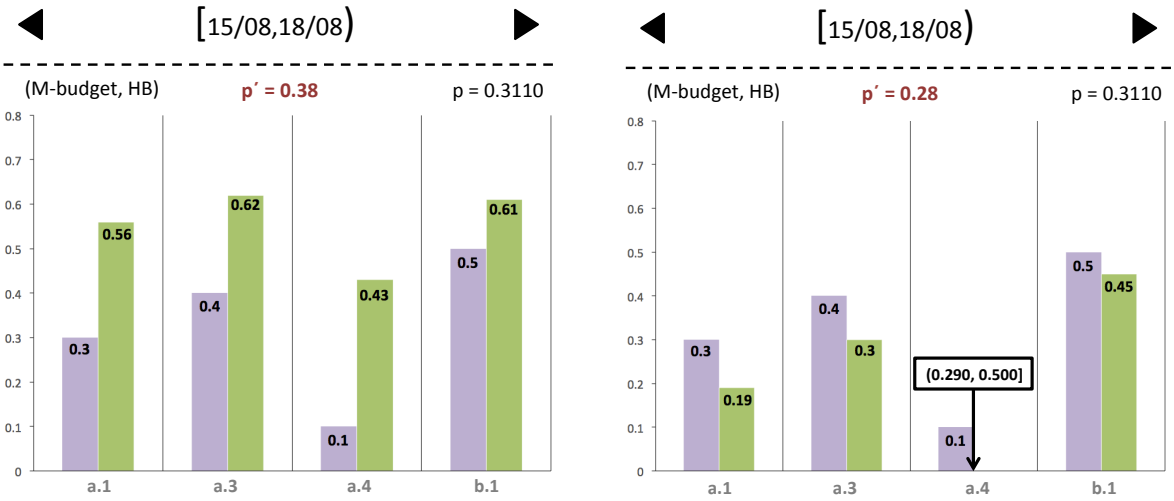


(a) Initial TPBC (b) TPBC with $b_1.p = 0.9$

Figure 4.3: The TPBC of q

The TPBC (Fig. 4.3a) of q illustrates the probability impact of a_1, a_3, a_4, b_1 on $q.p$. All the bubbles are light blue and, thus, an increase/decrease of the probabilities of the corresponding tuples will lead to an increase/decrease of $q.p$. The bubble for b_1 is positioned higher than the other ones, with $u_{b_1} = 0.622$, indicating that an increase of $b_1.p$ would lead to a greater increase in $q.p$ than the increase of $a_1.p, a_3.p, a_4.p$. Thus, the probability of tuple $q.p$ is mostly influenced by the probability with which *Ann shops in HB* (b_1).

In Fig. 4.3b, the TPBC of tuple q is illustrated for the case when $b1.p$ is increased by 0.4. The modification in $b1.p$ affected the probability impact of the other tuples in the TPBC but not the probability impact of b_1. The bubble for tuple a_3 is almost as high as the bubble for b_1, since $u_{a_3} = 0.567$ and $u_{b_1} = 0.622$. Although their height is similar, the bubble for b_1 has a greater diameter than the one for a_3 indicating that $a_3.p = 0.4 < b_1.p = 0.9$. Having a smaller value, $a_3.p$ proves more flexible to modifications and in such a case, it allows for $q.p$ to range over an interval with a higher maximum value. More specifically, by clicking on the bubble for a_3, the user is presented with the interval $(0.33, 0.9]$ over which $q.p$ will range when $a_3.p$ ranges over $(0, 1]$. On the contrary, the same interval for b_1 is $(0, 0.622]$.

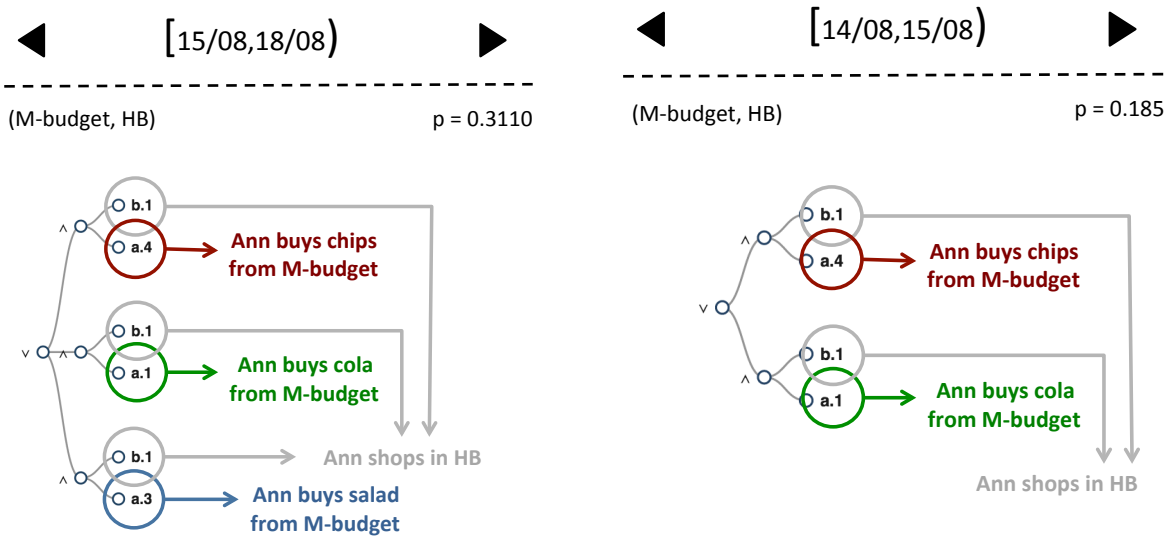


(a) $q.p' = 0.38$ (b) $q_{23}.p' = 0.28$

Figure 4.4: TPCC of q

With the focus being still on tuple q, if the user provides a new probability value $q.p' = 0.38$ for q, the TPCC (Fig. 4.4a) illustrates alternatives, involving the tuples a_1, a_3, a_4, b_1, in order to achieve this increase. For each of these tuples, the TPCC includes a purple column, with height equal to its initial probability, and a green column with a suggested probability value. The visualization of initial-suggested probabilities provides a perspective on the level of the values (high/low) as well as on the magnitude of their difference. One alternative is to increase $a_1.p$ by 0.26 whereas another involves increasing $a_3.p$ to 0.62. Similarly, if the user wishes that $q.p$ is decreased to $q.p' = 0.28$, the corresponding TPCC (Fig. 4.4b) shows that one alternative is to decrease $a_3.p$ to 0.3. In this case, tuple $a.4$ cannot contribute to such a modification, since it can only assist in the modification of $q.p$ within the interval $[0.290, 0.500]$.

In case the user needs a more detailed view of why the tuples a_1, a_3, a_4, b_1 have been studied under the goal of increasing $q.p$, the TPLT of this tuple is provided in Figure 4.5a. Using the arrows on top of the TPLT, similarly to the TPBC and TPCC, the user can navigate through the evolution of event $e = (M\text{-}budget, cola)$ over time. For example, the TPLT of tuple $q' = (M\text{-}budget, cola, [14/08, 15/08), 0.185)$ is illustrated in Fig. 4.5b. Although both tuples q' and q describe the same event, their TPLTs are different since they are not derived from the same base tuples at all time points. According to the TPLT of q, during *[15/08,18/08)*, the above event is influenced by the base tuple a_3, i.e., by Ann buying salad from M-budget, but a_3 is not present in the TPLT of q'.



(a) TPLT of q

(b) TPLT of q'

Figure 4.5: The TPLT

Aknowledgements

Special thanks to Peter Niederberger from Migros for the discussions and feedback.

CHAPTER 5

Conclusion and Future Work

In this thesis, we propose algorithms that efficiently compute the output intervals and lineage expressions of TP operations with negation: TP set-difference, TP outer-joins and TP anti-join and we introduce means for the understanding and analysis of their result. Firstly, we define temporal-probabilistic set-operations, including TP set-difference, and we proposed the *lineage-aware temporal window* for their computation. Its main characteristic is the binding of an output interval with the lineages of the input tuples that contribute to this interval at the time when it is formed. Exploiting this mechanism, we introduce an algorithm that produces the windows of two TP relations in *linearithmic time* that computes the result of a TP set operation after two simple filtering and lineage-concatenation steps on each window. Thus, we improve over existing quadratic approaches since we alleviate the expensive joins that are otherwise used to either perform the lineage-and-interval binding or to determine the output tuples.

For the computation of TP outer-joins and TP anti-join, we introduce *generalized lineage-aware temporal windows* that comply with the more demanding requirements of TP joins with negation: (a) input tuples of different non-temporal attributes can be combined, (b) multiple tuples of each input relation might be valid over an output interval and (c) more than one output tuples might be produced using the same input tuples at a time-point. We split the *generalized lineage-*

aware temporal windows of two TP relations into three disjoint sets and provide algorithms for the computation of these sets in an incremental manner. Pipelined computations allow for the integration of our approach in PostgreSQL, they alleviate the redundancies of existing approaches and guarantee but also as proven by an extensive experimental evaluation with real datasets.

Finally, we introduce three vizualization tools for the analysis of temporal-probabilistic results and we provide the user with the most important information to systematically modify the time-varying probability of the top-k result tuples of a TP query. By exploiting the linear relation between the probability of an output tuple and the probability of a base tuple included in its lineage expression, we illustrate the impact that a change in the probability of base tuples has on the output probability. As a result, the user easily identifies the input probabilities they should put more emphasis on. Our visualization tools are implemented in the form of a web application, on top of TP PostgreSQL implementation on which the users can submit their queries.

Future Work: Currently, we limit our approach to duplicate-free relations, i.e. there is a single prediction at each time-point and there are not two tuples that predict the probability of the same non-temporal attributes at a time-point. This assumption is widely adopted in both temporal and probabilistic databases. However, it would be interesting to investigate the impact that duplicates would have on the definition and implementation of temporal-probabilistic operations.

Furthermore, in this work we assume tuple independence, i.e. we assume that the probability of each base tuple at a time-point is dependent only on the tuple itself being true or false. In order to be able to model a wider range of application scenarios, we plan to investigate tuple correlations where the prediction related with the time-points in each tuple would be a conditional probability table rather than a single value. Correlations in temporal-probabilistic databases have never been investigated and pose challenges in the query evaluation as well as in establishing query semantics consistent with both dimensions.

Bibliography

[AGC+13] Mohammed Al-Kateb, Ahmad Ghazal, Alain Crolotte, Ramesh Bhashyam, Jaiprakash Chimanchode, and Sai Pavan Pakala. Temporal query processing in teradata. In *EDBT/ICDT*, pages 573–578, 2013.

[AKO07] Lyublena Antova, Christoph Koch, and Dan Olteanu. Maybms: Managing incomplete information with probabilistic world-set decompositions. In *ICDE*, 2007.

[All83] James F. Allen. Maintaining Knowledge About Temporal Intervals. *Commun. ACM*, 26(11):832–843, 1983.

[APR+98] Lars Arge, Octavian Procopiuc, Sridhar Ramaswamy, Torsten Suel, and Jeffrey Scott Vitter. Scalable sweeping-based spatial join. In *VLDB*, pages 570–581, 1998.

[B95] Michael H. Böhlen. Temporal database system implementations. *SIGMOD Rec.*, 24(4):53–60, December 1995.

[BBJ98] Michael H. Böhlen, Renato Busatto, and Christian S. Jensen. Point-versus interval-based temporal data models. In *Proceedings of the Fourteenth International Conference on Data Engineering, Orlando, Florida, USA, February 23-27, 1998*, pages 192–200, 1998.

[BDM+05] Jihad Boulos, Nilesh N. Dalvi, Bhushan Mandhani, Shobhit Mathur, Christopher Ré, and Dan Suciu. MYSTIQ: a system for finding more answers by using probabilities. In *SIGMOD*, 2005.

[BJ09] Michael H. Böhlen and Christian Jensen. Sequenced Semantics. In *Encyclopedia of Database Systems*, pages 2619–2621. Springer Berlin, Heidelberg, Germany, 2009.

[BJS00] Michael H. Böhlen, Christian S. Jensen, and Richard Thomas Snodgrass. Temporal statement modifiers. *ACM Trans. Database Syst.*, 25(4):407–456, December 2000.

[BSH+08] Omar Benjelloun, Anish Das Sarma, Alon Y. Halevy, Martin Theobald, and Jennifer Widom. Databases with uncertainty and lineage. *VLDB J.*, 17:243–264, 2008.

[CB17] Francesco Cafagna and Michael H. Böhlen. Disjoint interval partitioning. *VLDB J.*, 26(3):447–466, 2017.

[DBG12] Anton Dignös, Michael H. Böhlen, and Johann Gamper. Temporal alignment. In *SIGMOD*, pages 433–444, 2012.

[DBG14] Anton Dignös, Michael H. Böhlen, and Johann Gamper. Overlap interval partition join. In *SIGMOD*, pages 1459–1470, 2014.

[DBGJ16a] Anton Dignös, Michael H. Böhlen, Johann Gamper, and Christian S. Jensen. Extending the kernel of a relational dbms with comprehensive support for sequenced temporal queries. *ACM Trans. Database Syst.*, 41(4):26:1–26:46, November 2016.

[DBGJ16b] Anton Dignös, Michael H Böhlen, Johann Gamper, and Christian S Jensen. Extending the Kernel of a Relational DBMS with Comprehensive Support for Sequenced Temporal Queries. *TODS*, 41(4):26:1–26:46, 2016.

[DMT13] Maximilian Dylla, Iris Miliaraki, and Martin Theobald. A temporal-probabilistic database model for information extraction. *PVLDB*, 6(14):1810–1821, 2013.

[DRS01] Alex Dekhtyar, Robert Ross, and V. S. Subrahmanian. Probabilistic temporal databases, i: Algebra. *ACM Trans. Database Syst.*, 26(1):41–95, 2001.

[DS07] Nilesh N. Dalvi and Dan Suciu. Efficient query evaluation on probabilistic databases. *VLDB J.*, 16(4):523–544, 2007.

[DS12] Nilesh N. Dalvi and Dan Suciu. The dichotomy of probabilistic inference for unions of conjunctive queries. *J. ACM*, 59(6):30:1–30:87, 2012.

[FHO13] Robert Fink, Jiewen Huang, and Dan Olteanu. Anytime approximation in probabilistic databases. *VLDB J.*, 22(6):823–848, 2013.

[FO11] Robert Fink and Dan Olteanu. On the optimal approximation of queries using tractable propositional languages. In *ICDT*, pages 174–185, 2011.

[FO16] Robert Fink and Dan Olteanu. Dichotomies for queries with negation in probabilistic databases. *ACM Trans. Database Syst.*, 41:4:1–4:47, 2016.

[FOR11] Robert Fink, Dan Olteanu, and Swaroop Rath. Providing support for full relational algebra in probabilistic databases. In *ICDE*, pages 315–326, 2011.

[GS14] Wolfgang Gatterbauer and Dan Suciu. Oblivious bounds on the probability of boolean functions. *TODS*, 39(1):5, 2014.

[GS15] Wolfgang Gatterbauer and Dan Suciu. Approximate lifted inference with probabilistic databases. *PVLDB*, 8(5):629–640, 2015.

[GS17] Wolfgang Gatterbauer and Dan Suciu. Dissociation and propagation for approximate lifted inference with standard relational database management systems. *VLDB J.*, 26(1):5–30, 2017.

[IL84] Tomasz Imieliński and Witold Lipski, Jr. Incomplete information in relational databases. *J. ACM*, 31:761–791, 1984.

[Jen00] Christian S Jensen. Introduction to temporal database research. *Temporal database management*, 2000.

[JOS10] Abhay Jha, Dan Olteanu, and Dan Suciu. Bridging the gap between intensional and extensional query evaluation in probabilistic databases. *Proceedings of the 13th International Conference on Extending Database Technology - EDBT '10*, page 323, 2010.

[KLD11] Bhargav Kanagal, Jian Li, and Amol Deshpande. Sensitivity analysis and explanations for robust query evaluation in probabilistic databases. In *SIGMOD*, 2011.

[KLS+15] Zuhair Khayyat, William Lucia, Meghna Singh, Mourad Ouzzani, Paolo Papotti, Jorge-Arnulfo Quiané-Ruiz, Nan Tang, and Panos Kalnis. Lightning fast and space efficient inequality joins. *PVLDB*, 8(13):2074–2085, 2015.

[KMV+13] Martin Kaufmann, Amin Amiri Manjili, Panagiotis Vagenas, Peter M. Fischer, Donald Kossmann, Franz Färber, and Norman May. Timeline index: a unified data structure for processing queries on temporal data in SAP HANA. In *SIGMOD*, pages 1173–1184, 2013.

[Koc08] Christoph Koch. On query algebras for probabilistic databases. *SIGMOD Record*, 37(4):78–85, 2008.

[KRT11] Sanjeev Khanna, Sudeepa Roy, and Val Tannen. Queries with difference on probabilistic databases. *PVLDB*, 4(11):1051–1062, 2011.

[KVF+13] Martin Kaufmann, Panagiotis Vagenas, Peter M. Fischer, Donald Kossmann, and Franz Färber. Comprehensive and interactive temporal query processing with SAP HANA. *PVLDB*, 6(12):1210–1213, 2013.

[LM97] Nikos A Lorentzos and Yannis G Mitsopoulos. SQL extension for interval data. *TKDE*, 9(3):480–499, 1997.

[MTea07] Michi Mutsuzaki, Martin Theobald, and et al. Trio-one: Layering uncertainty and lineage on a conventional DBMS (demo). In *CIDR*, 2007.

[OH08] Dan Olteanu and Jiewen Huang. Using OBDDs for efficient query evaluation on probabilistic databases. In *SUM*, pages 326–340, 2008.

[OHK09] Dan Olteanu, Jiewen Huang, and Christoph Koch. Sprout: Lazy vs. eager query plans for tuple-independent probabilistic databases. In *ICDE*, pages 640–651, 2009.

[OHK10] Dan Olteanu, Jiewen Huang, and Christoph Koch. Approximate confidence computation in probabilistic databases. In *ICDE*, pages 145–156, 2010.

[PHD16] Danila Piatov, Sven Helmer, and Anton Dignös. An interval join optimized for modern hardware. In *ICDE*, pages 1098–1109, 2016.

[PTB18] Katerina Papaioannou, Martin Theobald, and Michael Böhlen. Set operations in temporal-probabilistic databases. In *ICDE*, 2018.

[RDS07] Christopher Re, Nilesh N. Dalvi, and Dan Suciu. Efficient top-k query evaluation on probabilistic data. In *ICDE*, 2007.

[SOCK11] Dan Suciu, Dan Olteanu, R. Christopher, and Christoph Koch. *Probabilistic Databases*. Morgan & Claypool Publishers, 1st edition, 2011.

[STW08] Anish Das Sarma, Martin Theobald, and Jennifer Widom. Exploiting lineage for confidence computation in uncertain and probabilistic databases. In *ICDE*, pages 1023–1032, 2008.

[Suc09] Dan Suciu. Probabilistic Databases. In *Encyclopedia of Database Systems*, pages 2150–2155. Springer Berlin, Heidelberg, Germany, 2009.

[Tom98] David Toman. Point-based temporal extensions of SQL and their efficient implementation. In *Temporal databases: research and practice*, pages 211–237. Springer, 1998.

[VL07] Jose R. Rios Viqueira and Nikos A. Lorentzos. SQL Extension for Spatio-temporal Data. *VLDB-J*, 16(2):179–200, 2007.

[Wid09] Jennifer Widom. Trio: A system for data, uncertainty, and lineage. In Charu C. Aggarwal, editor, *Managing and Mining Uncertain Data*, chapter 5, pages 113–148. Springer, 2009.

[WKL+08] Evan Welbourne, Nodira Khoussainova, Julie Letchner, Yang Li, Magdalena Balazinska, Gaetano Borriello, and Dan Suciu. Cascadia: a system for specifying, detecting, and managing RFID events. In *MobiSys*, pages 281–294, 2008.

[WRS08] Ting-You Wang, Christopher Re, and Dan Suciu. Implementing not exists predicates over a probabilistic database. In *QDB/MUD*, pages 73–86, 2008.